CSE 312

OPERATING SYSTEMS

Midterm Report

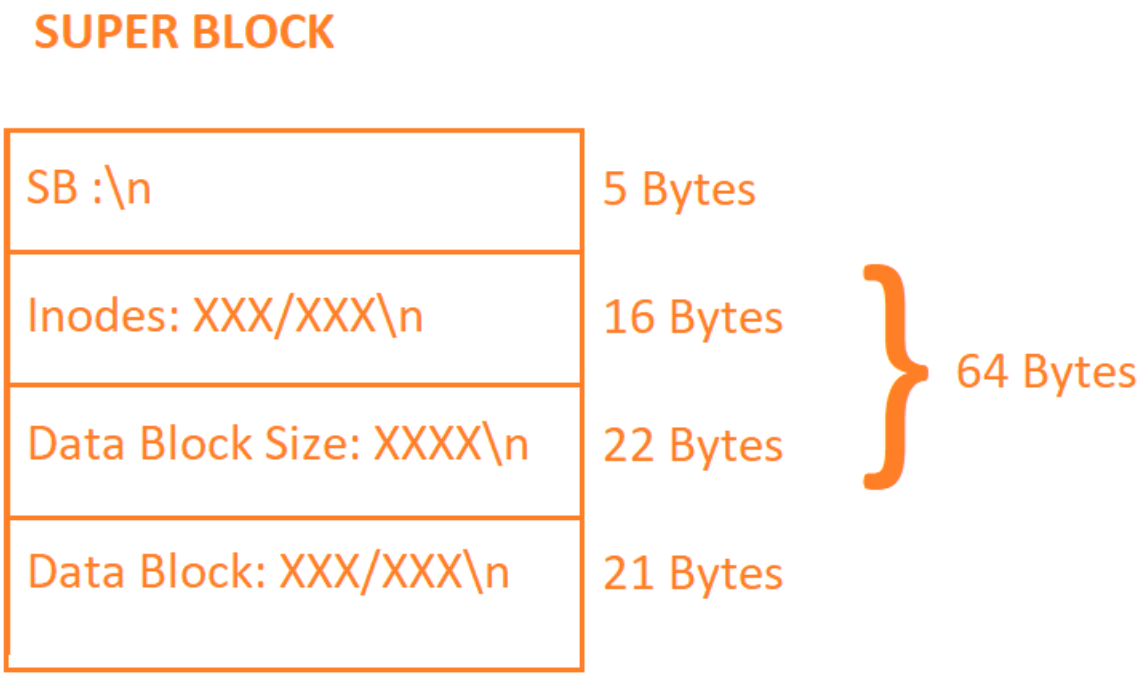
Ahmet Yuşa Telli 151044092
31.05.2020

1. STRUCTURE DESIGN

In our memory, we have 5 different types blocks: super block, root block, i-node block, i-nodes, data block.

a. Super Block:

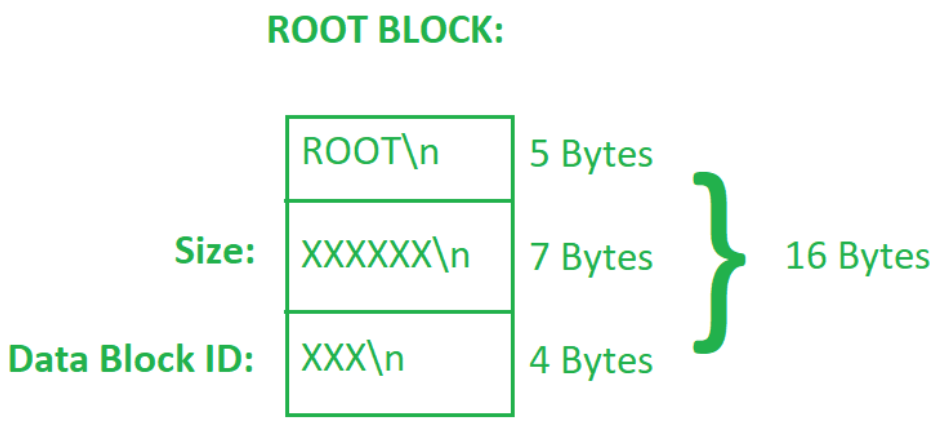
We have one super block on top of our memory. Its size is 64 bytes. We store free inodes, total inodes and free data blocks, total data blocks and one data block size.



b. Root Block

We have a root block for store files. When we create a file, we write its information under Root's data block. The ID of the data block of the Root Block is **1**. When we take "mkdir" operation we write the first file under the root data block with ID **1**.

We have a title "ROOT". This block should be unique. We store root directory's size with six-digit number. And its data blocks ID with three-digit number. The Root block takes up 16 bytes of memory.



c. Inode Block

This block saves our inodes. We get the number of inodes and size of inode from the user. Then we calculate how many inodes in one inode block. For example: we get 4KB inode block size and 400 inodes from the user. One inode block size is 4 * 1024 = 4096 bytes. Our inode's size is 64 bytes. Therefore, each inode block has 4096 / 64 = 64 inodes.

Then we need to find how many inode blocks we should create. 400 / 64 = 6,25. Ops. We need 7 inode blocks but the last inode block will not be full. In 6 inode block we store 6 * 64 = 384 inodes. But we have 400 inodes. The last inode block has only 16 inodes.

In part 2, we calculate all these things and we create a 1 MB file for store these.

i. Inode Structure

We get number of inodes from the user. Our inode store some information:

- ID
- Valid
- Size
- Direct Block 1
- Direct Block 2
- Direct Block 3
- Direct Block 4
- Undirect Block 1

ID numbers are unique for each inodes. Valid is T/F. If the inodes is used before its valid is F.

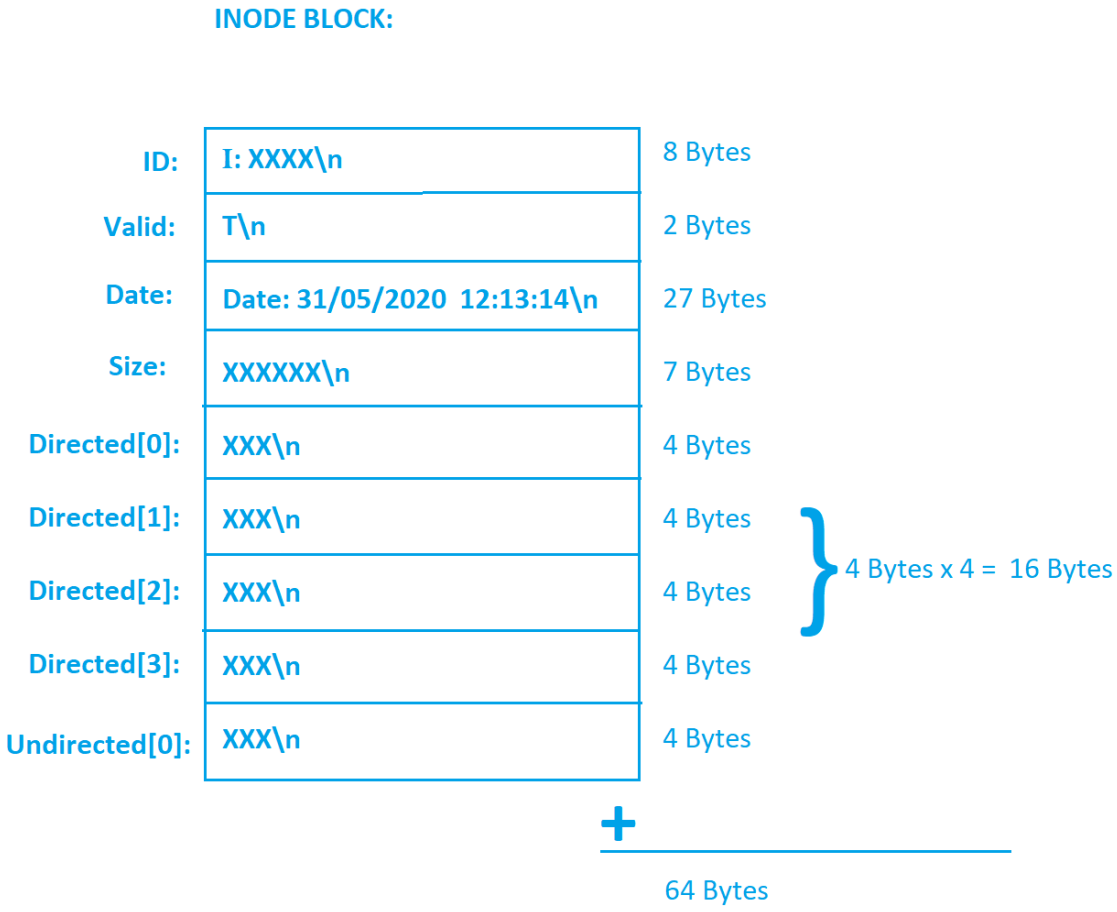
When we create inodes its size is 0. If it gets some data from user, we update this inodes size. Size is six-digit number.

One inode has 4 data block and 1 undirect data block. When we create an inode, it has not any data blocks. All inodes data blocks ID are 0. If one data block is full. We give a new data block. After 4 data blocks are full. We give a undirect data block.

Undirect data block stores only data blocks ID. It does not store any data.

Therefore, one inode can store 4 * (data block size) with direct data blocks. If we say one data block is 4096 bytes, 4 * 4096 = 16384 bytes = 16KB stores with direct data block. For more memory we use undirect data block. One undirect data block size is as much as the data block. But is stores data blocks ID. One data block ID is three-digit number. It is 4 bytes with newline character. One undirect block can store 4096 / 4 = 1024 data block ID. It means, we can use 1024 data block with one undirect data block. Therefore, one undirect block can store 1024 * 4096 = 4,194,304 bytes, so 1024 * 4KB = 4096KB = 4MB. One undirect block can store 4MB data.

As a result of all these calculations, one inode can store 4MB + 16KB.

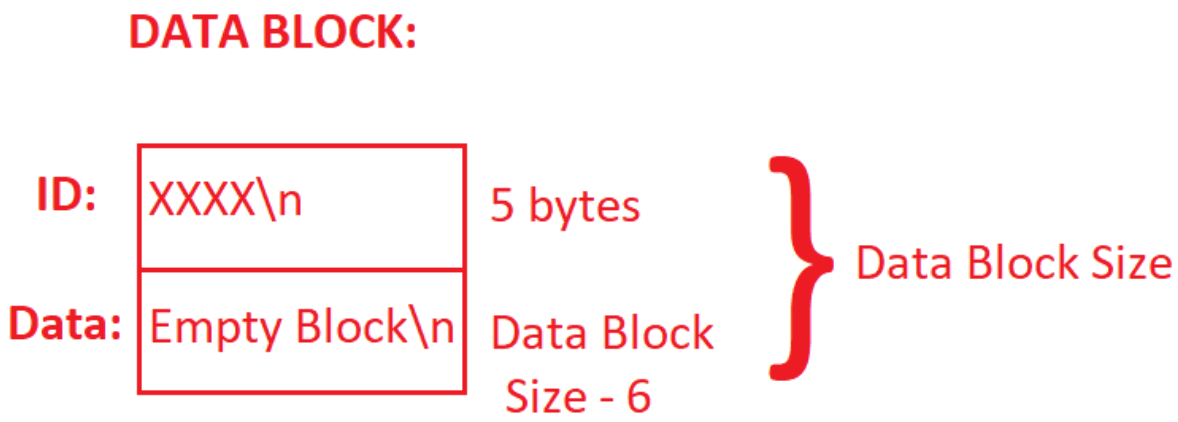


d. Data Block

We store data using data block. Its size we get from the user. Only we give four-digit number ID. Our file system is 1MB. To find how many data block we have, we must first calculate the remaining area. Let's say we have 4KB data block size and 400 inodes.

We have one super block, one root block, 400 inodes. Their place in memory is: 64 bytes + 16 bytes + 400 * 64 bytes = 25,680 bytes. (1MB = 1,048,576 bytes.)

We have 1,048,576 – 25,680 = 1,022,896 bytes for data blocks. One data blocks size is 4096. We have 1,022,896 / 4096 = 249 data blocks.



2. Free Blocks

When we create a file system, we write number of inodes and free inodes, number of data blocks and free data blocks and size of data blocks to the super block.

When we are doing file operations, we read free inodes and free data blocks from file system. After operations, we update super block information, write to the file system.

3. Part 3 Definition

a. Functions

- void get_date(char * d);
  This function gives us the current time. We use for update inodes.
- int get_data_block_id();
  This function gives us a new data block id. At the beginning we write 0, then we take new ID.
- int update_inode();
  When we make a new file, we give an inode to new file. And update its information; size, create time, date and data block ID.
- int uygun_inode_bul();
  This function checks inodes valid bits and gives us an unused inode ID.
- int check_file_oper(const char * str1, const char* str2);
  This funcition checks input operations and checks if two strings are the same.
- void make_mkdir(char * filename);
  When the mkdir operations is an input, we call this function first.
- int check_root_dir(char * str);
  When we write something to the file system, we start to check at root directory. And we check the first file in the input, did we create before this. We check this. If the first file created before, we return its inode ID.
- void inode_a_yaz(char* c,int id);
  When we add a new file under the previously created file. We take child file's name and we take a new inode ID. Then we write new child file under the its inode's data block.
- void update_root_dir(int i, char * s);
  When we add a new file under the root directory. We call this.
- void make_rmdir(char * filename);
  The rmdir operation we make it in this function.
- void get_inode_info(int id);
  When we make ls operation, we take inodes information. It gives us date and size.
- void make_list_l();
  The ls-l operation we call this. Print inode ID, size, date, filename.
- void make_read(char * dest, char * target);
  The read operation we call this.
- void make_write(char * dest, char * target);
  The write operation we call this.
- void tokenize(char * target);
  When we write a file to the file system, we take the file's names without ",",/.
- int find_file_id(char * str);
  Search a filename in data block and return its data block ID

- int find_db_id(int a);
  In “a” inode, return its data block ID.
- int update_inode_size(int a, int b);
  When we write some information to data block and we should update its inode and inode’s size.
- int delete_inode(int a);
  When we make rmdir or del, we should remove inode information.
- int delete_data_block(int a);
  We delete a data block and make a new data block.
- void update_super_block(char * filename);
  After running this program, we need to update super block and write new free blocks and free data blocks.
- void make_dumpe();
  The dumpe2fs operation is as input, we call this. List free inodes and data blocks numbers.
- int update_file_2(char * c);
  When we make mkdir, we check all special things and we write carefully this file to the file system.
- void update_root_block(int i);
  We update root block; we read and write number of root directory files.
- int check_inode_dir(int id, char * str);

  When we take more files, we should check parent files created or not. If parents created previously, we take its inode ID and go to the its data block. Then, check the next file in this data block.

b. Operations

i. MKDIR

When mkdir operation comes from the user, first we tokenize the input take files names. First, we count how many files names came. Then, check first file name is created before in root directory. If it created, we take its inode ID. And second file we add this inode's data block.

If one file came, we write it in root directory. And we give a new inode ID and data block ID to it. Then we update root block.

ii. RMDIR

First, we tokenize the files names. Then we take the last file's name. After, go to the data block and check its inode ID. We go to this inode and remove its information size, date and data blocks ID's. Then go to the data blocks and clear them too. We do not check it is empty or not.

Note: We remove only last file. We do not check it is empty or not.

iii. LIST

This operation works for only root directory. It works like "ls -l". First, we go to the root directory and take files inode ID's and their names, then we go to its inodes and write its date and size. Output is:

```
cse312@ubuntu:~/Desktop/part3$ ./exe
fileSystemOper fileSystem.data list "/"
fileSystemOper fileSystem.data list "/"
File system name:fileSystem.data
1       0 31/ 5/2020   8:32:41 os
2       0 31/ 5/2020   8:32:51 usr
```

iv. DUMPE2FS

I did not fully understand what to do. I list number of total and free inodes and number of total and free data blocks. Then, I list files information. Output is:

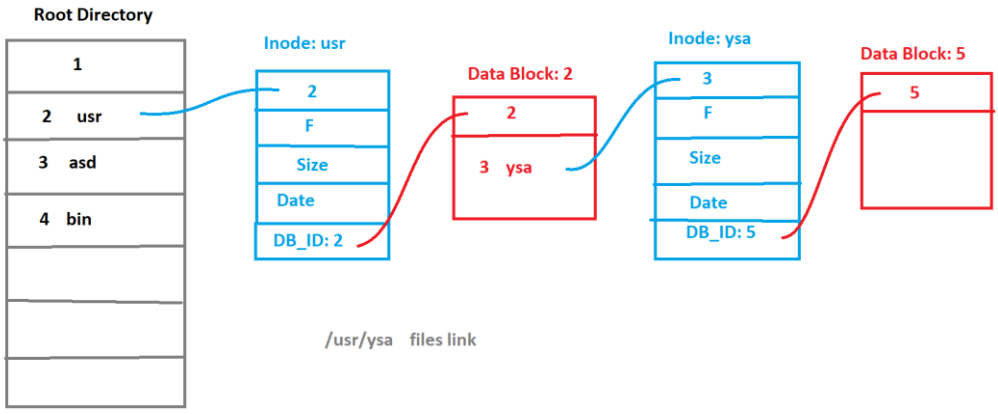
```
cse312@ubuntu:~/Desktop/part3$ ./exe
fileSystemOper fileSystem.data dumpe2fs
fileSystemOper fileSystem.data dumpe2fs
File system name:fileSystem.data
dumpe2fs operation
Total Inode Blocks: 400
Free  Inode Blocks: 396
Total  Data Blocks: 249
Free    Data Blocks: 245
1       0 31/ 5/2020   8:32:41 os
2       0 31/ 5/2020   8:32:51 usr
```

v. WRITE

We take two files names from the user. The input should be like this:

```
fileSystemOper fileSystem.data write "/usr/ysa" linuxFile.data
```

First, we read `linuxFile.data` file. Take its information and write to `"/usr/ysa"` 's data block. We write its information in to the data block with ID 5. Here is what I did:



vi. READ

We take the names of the two files from the user. The input should be like:

```
fileSystemOper fileSystem.data read "/usr/ysa" linuxFile2.data
```

First, we go to the "ysa" (as in the previous picture) data block and take its information. Then, we write in to the `linuxFile2.data` file.

Important Note: "Write" and "Read" operations may be mixed. I understood that this should be done.

vii. DEL

This operation works very similar to "rmdir" operation. It takes a file name and go to its inode and clear inode. Then go to the its data block and clear data blocks.

4. HOW TO RUN

We have three different parts in this midterm. The second part, I talked about my design in first title. I mentioned the third part in the third title. I made the second and third parties separately. There are two different makefiles, main.cpp files. But there is same fileSystem.data. When you test my midterm, first you should run part 2' make file. Then, run it using "**./exe**" . You should check comment.txt file. Copy the first line and paste to the terminal. **Part 2 only create a "file system" file.** Then copy the .data file to the part 3 folder.

Go to part 3 file. First run make file. Then run using "**./exe**" . You can use comment.txt file for test. You should copy one line and paste to the terminal line by line.

You should check readme.txt file.

Ahmet Yuşa Telli

151044092