Project Report

Group No- 1

File System

Anant Sharma-Y13uc025
Avinash Ankur- Y13uc058
Gopal Kumar- Y13uc105
Govinda Raj- Y13uc106
Kashish Raheja- Y13uc139

We have created a new file system to understand concept of file system like ext2, ext3, NTFS, JFS, fat32. So we are simulating one in user space as follows:

- Using a regular storage for a partition, that is for the actual data storage of the file system.
- Designing a basic file system structure and implementing the format for it, over the regular storage.
- Providing an interface/shell to type commands and operate on the file system, similar to the usual bash shell. In general, this step is achieved by the shell along with the corresponding file system driver in kernel space. But here that translation is simulated into the user space interface, itself.

Properties of file system-

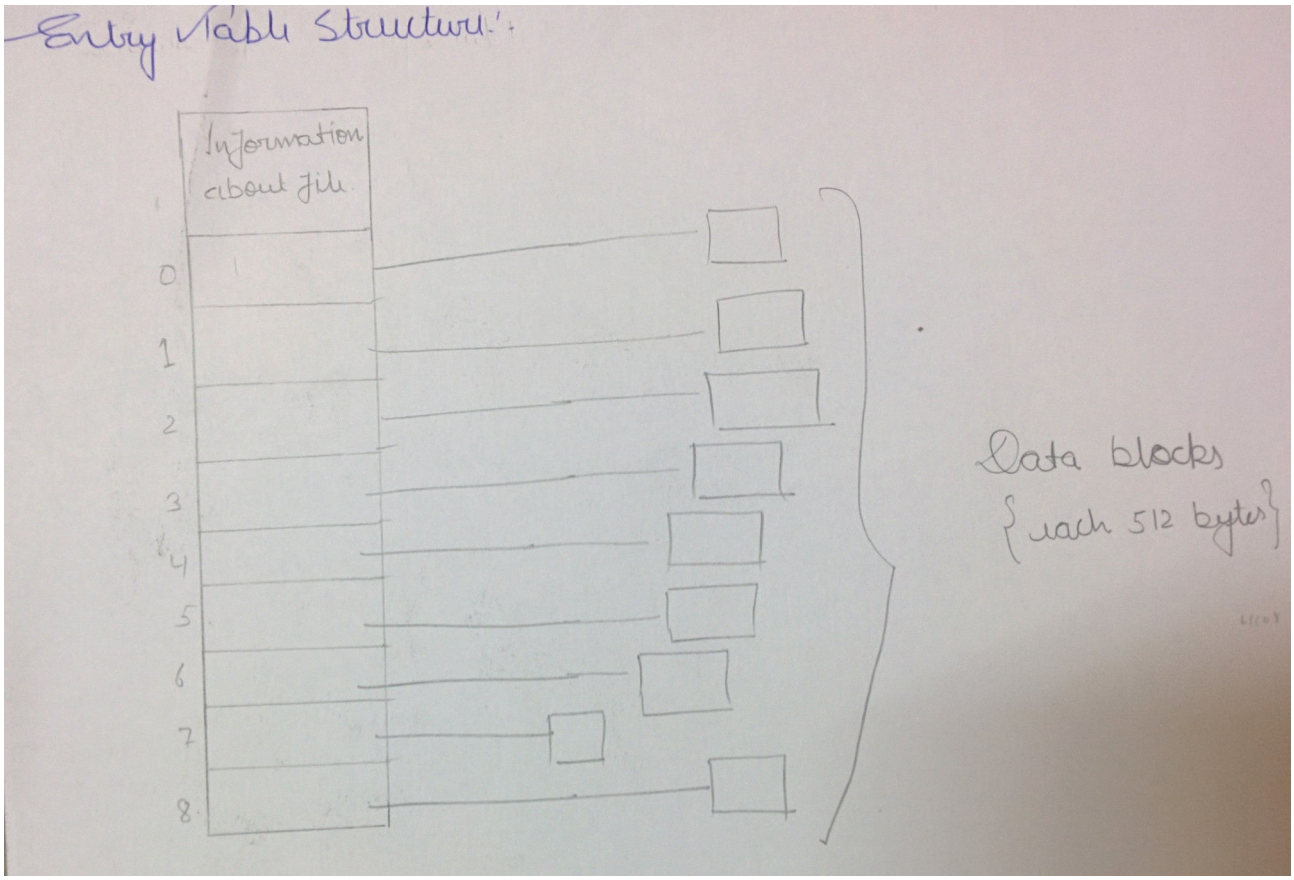
- Block Size- 512bytes
- Entry size- 64bytes
- Magic number- 0x13090D15 (Any random hexadecimal number)

We are using superblock to store following information about file system.

- Magic number to identify the file system.
- Block size
- Partition size
- Entry size
- Entry table size
- Entry table block start
- Entry count
- Data block start

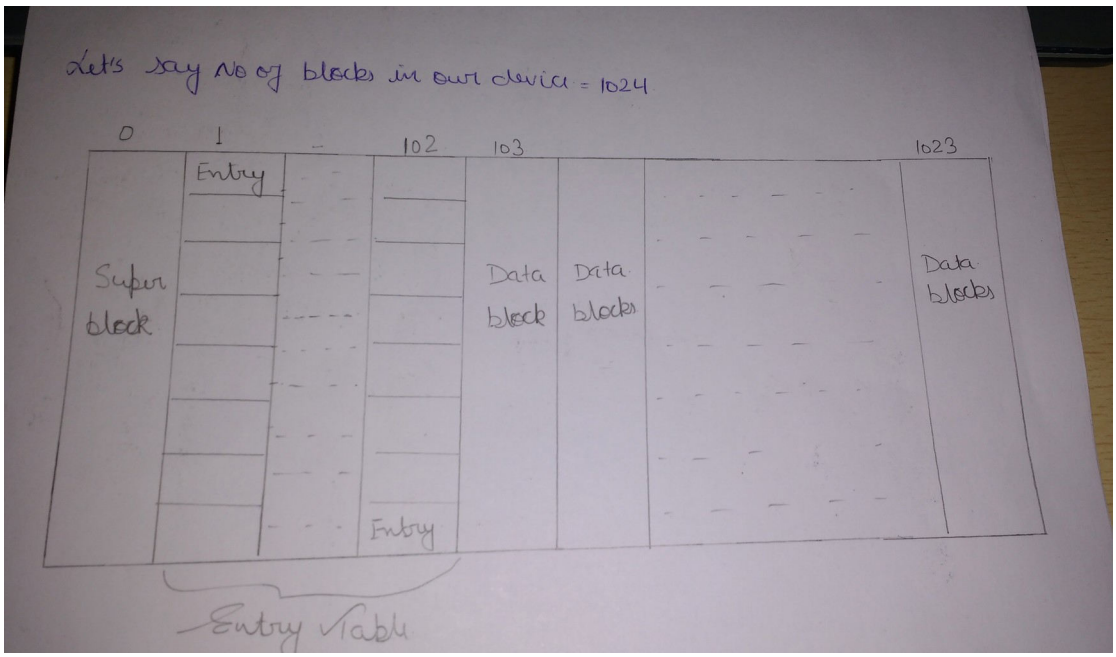
We are using entry table to store following information about each file.

- Name of up to 15 character(1 byte for “)
- Size in bytes
- Time stamp (creation or modification)
- Permissions (just one set, for the user)
- Array of block numbers for up to 9 data blocks.



The heuristic of 10% of total space to be used for the file entries, defined by SFS_ENTRY_RATIO.

- Its first block as the super block.
- The next 10% of total blocks as the file entry’s zeroed out table.
- And the remaining as the data blocks.



We have created one header and two C files. Header file which contains properties of file system in “superblock” and entry table which contains properties of each file. First C file is formating the partition into our file system. It is first clearing whole partition, which will be completely blank. After this it is changing the partition into our format of file system. The second C file is providing an interface to type following commands and operate on the file system :

- list- It lists all the file in that file system.
- create <file name> - It creates file with given file name.
- remove <file name> - It removes the specified file.
- write <file name> - It will allow to write into the given file.
- read <file name> - It will display the content of the given file.
- ? - Lists all the commands supporting our interface.
- quit – It will exit from interface.
- chperm <0-7> <file name> - It will change the permission of the file according to the given number.
  - 0- No permission
  - 1- Only execute
  - 2- Only write
  - 3- Write and execute
  - 4- Only read
  - 5- Read and execute
  - 6- Read and write
  - 7- Read, write and execute.

If we enter any other command or misspelled command then it will show proper error message and display all the supported commands.

Create a file

As we provide a file name to create a file, it will first traverse the entry table so that the file with provided name must not be present there. If in case it happens it will show error with that file name. Our default permission value is 7 for every newly created file that means in starting it will have all three 'write, read and execute permission'. Later on we can change permission of file.

Listing a file

As we type the list command, it will list out all the available file by traversing entry table with file name, size, permissions, and its creation time.

Removing a file

When we want to remove any file by providing the file name, first it will check whether file with provided name exist or not, if in case it doesn't exist it will show error messege with specified file doesn't exist. Otherwise it will get pointer of data block and will set all value of data block to the zero so that, that block will be used again by deleting information of specified file from entry table.

Reading a file

Reading a file is basically sequentially reading & displaying the contents of the data blocks indicated by their position from the blocks array of file's entry and displaying that on stdout's file descriptor 1. A couple of things to be taken care of:

- File is assumed to be without holes, i.e. block position of 0 in the blocks array indicates no more data block's for the file.
- Reading should not go beyond the file size. Special care to be taken while reading the last block with data, as it may be partially valid.

Quitting from interface

To quit from the interface we have to simply write the command 'quit' and it will quit from the interface by terminating the program.

Writing a file

We will search all free available blocks, which has to be obtained, filled and then their position be noted sequentially in the blocks array of the file's entry. Typically, we do this whenever we have received a block full data, except the last block. To know the last block we read till end of input, marked by 'Control-D' on its own line

from the user – and that is indicated by a return of 0 from read. And in that case, we check if any non-full block of data is left to be written, and if yes we follow the same procedure of obtaining a free available block, filling it up (with the partial data), and updating its position in the *blocks* array.

After all this, we have finally written the file data, along with the data block positions in the *blocks* array of the file's entry. And now it's time to update file's entry with the total size of data written, as well as timestamp to currently modified. Once done, this entry has to be updated back into the entry table, which is the last step. And in this flow, the following shouldn't be missed out during getting & filling up free blocks:

- Check for no more block positions available in blocks array of the file's entry.
- Check for no more free blocks available in the file system.

Data can be stored partially in block, means one block can contain more than one file and one file may be stored in more than one block. It has been achieved by doing offsets, which will show position of data in that particular block.

Change permission of file

To change the permission of pre-existed file we have to write command

'chperm <0-7> <file name>". This command will existence of provided file, if that file doesn't exist it will simply show error message. Otherwise it will change the permission of that file by looking on provided integer. It can be achieved by changing information of permission in entry table.

Integer	Action
0	No permission
1	Only execution
2	Only write
3	Write and execution
4	Only read
5	Read and execution
6	Read and write
7	Read, write and execution

Version 1.0

In version 1.0, we made only format file which was formatting our file system through command in provided interface. In this version we provided just the create, list and remove the file option in the interface.

Version 2.0

In version 2.0 we added initial version of writing the file and reading the file.

Version 3.0

This version contains one major update which is changing permission of file.

Version 4.0

In this version we improved 'write' command by doing some experiment to provide solution for internal fragmentation and code refactoring was done.

References:

- https://en.wikipedia.org/wiki/Ext2
- https://www.kernel.org/doc/Documentation/filesystems/ext3.txt
- http://codewiki.wikidot.com/c:system-calls:lseek
- http://comsci.li HYPERLINK "http://comsci.liu.edu/~murali/unix/write.htm"u.edu/~murali/unix/write.htm
- http://comsci.liu.edu/~murali/unix/read.htm
- http://codewiki.wikidot.com/c:system-calls:open
- http://man7.org/linux/man-pages/man2/openat.2.html
- http://www.java-samples.com/showtutorial.php?tutorialid=591