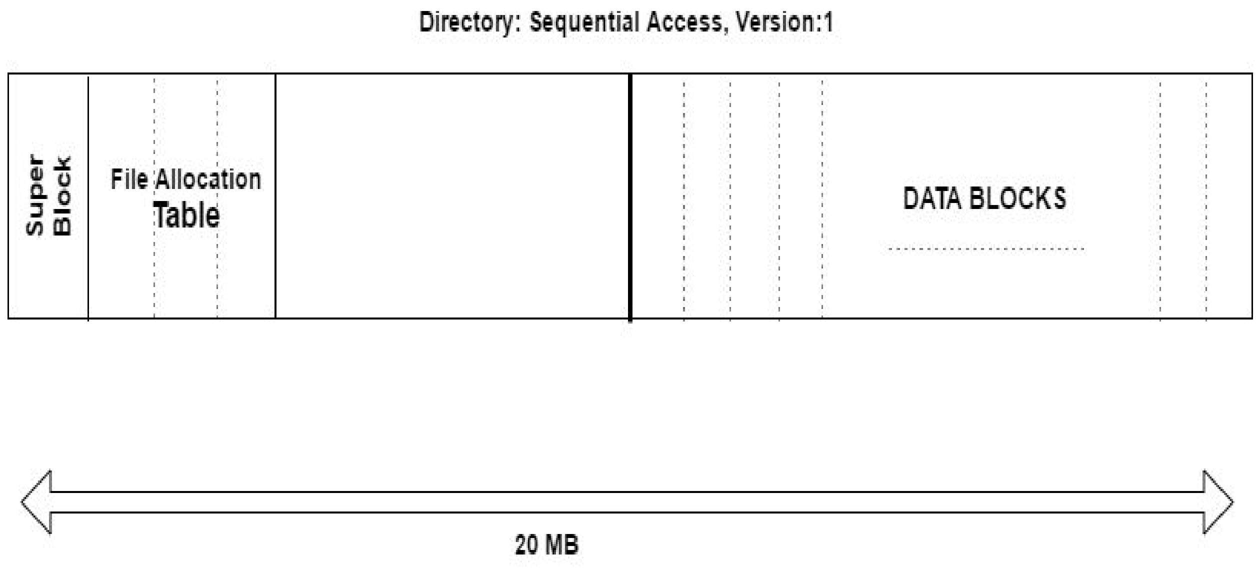
Project Report

"Efficient File Directory Implementation"

Group Members:
Mahima Achhpal (201301199)
Deeksha Koul (201301435)

Guide: Prof. Naresh Jotwani
Subject: IT 308 : Operating Systems

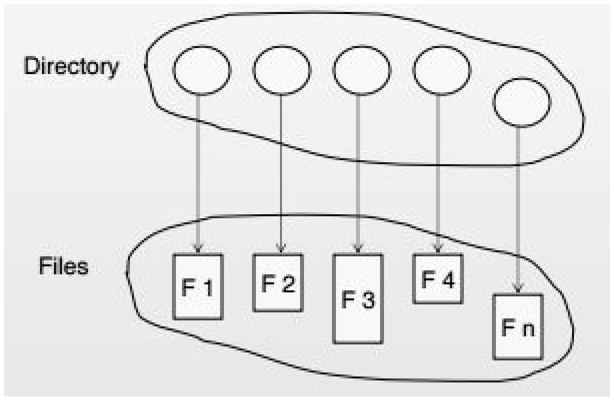
Basic File implementation done in Version_1.0 :



Basic file structure layout is seen in the above figure.File systems are stored on disks.In our version we have consider the super block to occupy the first block (no need of boot menu as we are already creating virtual Linux file), whereas the disk has been divided into blocks. For a 20MB space, the number of total blocks are 5120 given that the size of each block is 4KB. In our code version, we consider half of space is assigned to data blocks that will contain all files that we can read or write.The first half of net space will be comprise of super block(0th block) , FAT table(1st,2nd,3rd block) , Directory Information(the blocks after FAT , where information of each directory is stored).The file allocation table (FAT) is convenient because it can be used to keep track of empty blocks and the mapping between files and their data blocks.

For creating and accessing the Linux virtual disk of 20 MB, we had provided definitions and helper functions that will be present in the header file 'header.h' and the source file ' vfs.c' . In our code we have to use the helper functions and source file for the storage of all the data on our own created virtual disk.Our code gives the ability to user to create an empty disk, open and close a disk, and read and write entire blocks (by providing a block number in the range between 0 and 5,119 inclusive).

In version_1.0, we represented the basic file system with a one step directory implementation i.e directories are accessed in a sequential order as shown below.



We define the structures and macros for our version as follows:

Structures for:

a. Super block (3 parts)
b. directory entry
c. file descriptor

Global Variables :

a. struct super_block sb;
b. int *FAT;
c. struct dir_entry *DIR ;
d. int entries = 0; no of entries in directory

Macros Defined (#define ‘ ‘):

a. MAX_F_NAME 32 **:** MAximum characters possible in file name
b. DATA_BLOCKS 2560 **:** Data Blocks size in terms of number of blocks
c. MAX_FAT_SIZE 3 **:** (no. of data blocks *size of int) /BLOCK_SIZE = 2.5 blocks

To manage our file system,we have coded and utilized the following three functions:

1. void make_fs(char *);
2. void init_super_block();
3. void mount_fs(char *);
4. int fs_create(char *name);
5. struct file_descriptor* fs_open(char *name);
6. int fs_delete(char *name);
7. int get_free_block();

8. int dir_create_file(char * name);
9. struct dir_entry* dir_list_file(char * name);
10. void print_dir();
11. void print_FAT();
12. void load_Directory();
13. void load_FAT();
14. void write_back_Directory();
15. void write_back_FAT();
16. void unmount_fs(char *);
17. int fs_write(char * name, struct file_descriptor* fd, void *src, int nbytes);
18. int fs_read(char * name, struct file_descriptor* fd, void *dest, int nbytes);

"make_fs" function will build an empty file system on our virtual disk. Within this function, we first invoke make_disk('MyDisk') to create a new disk. Then, open_disk and initialize the given meta-information for the file system so that it can be later used,this is known as mounting of the disk(mousnt_fs). The function returns 0 on success, and -1 if the disk 'MyDisk' could not be created, opened or initialized.

After the mount operation, the file system becomes operational,all the meta-information gets loaded in the disk (vfs.c) so that all the other file system operations can be used now.

Unmounting the file system(unmount_fs) from a virtual disk, we needed to write back all meta-information so that the disk always is in sync with the changes that had been made to the file system (such as creating new files read/write files). After Unmounting,all data(that was temporarily present in memory) must be written onto the virtual disk.

For deleting or opening a file we need to first search the file this is accomplished by function dir_entry* dir_list_file(char * name); whereas after a file gets deleted content of the block it belongs changes and hence the blocks assigned to it becomes free.Through the method get_free_block(); we can get free blocks and assign them to the newly created directory.

Other than these basic functions, the rest are the simple filesystem implementation functions such as open/close, read/write , delete and create .

printFAT():prints the fat table at any instant of time

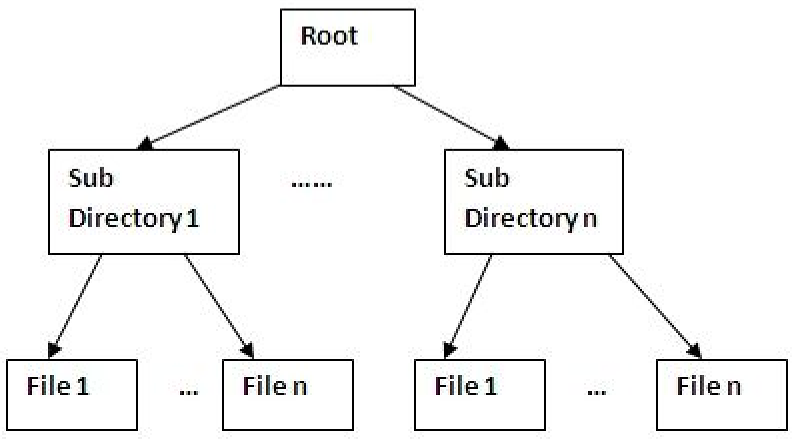
Load_dir():loading the directory entries in memory allocated.

Load_FAT():Initialise FAT and enter the information of files that have been created(the blocks numbers assigned to them) print_dir():print all the entries of a directory.

write_back_FAT/directory:Writes back to disc, works similar to unmounting.

Version 2:

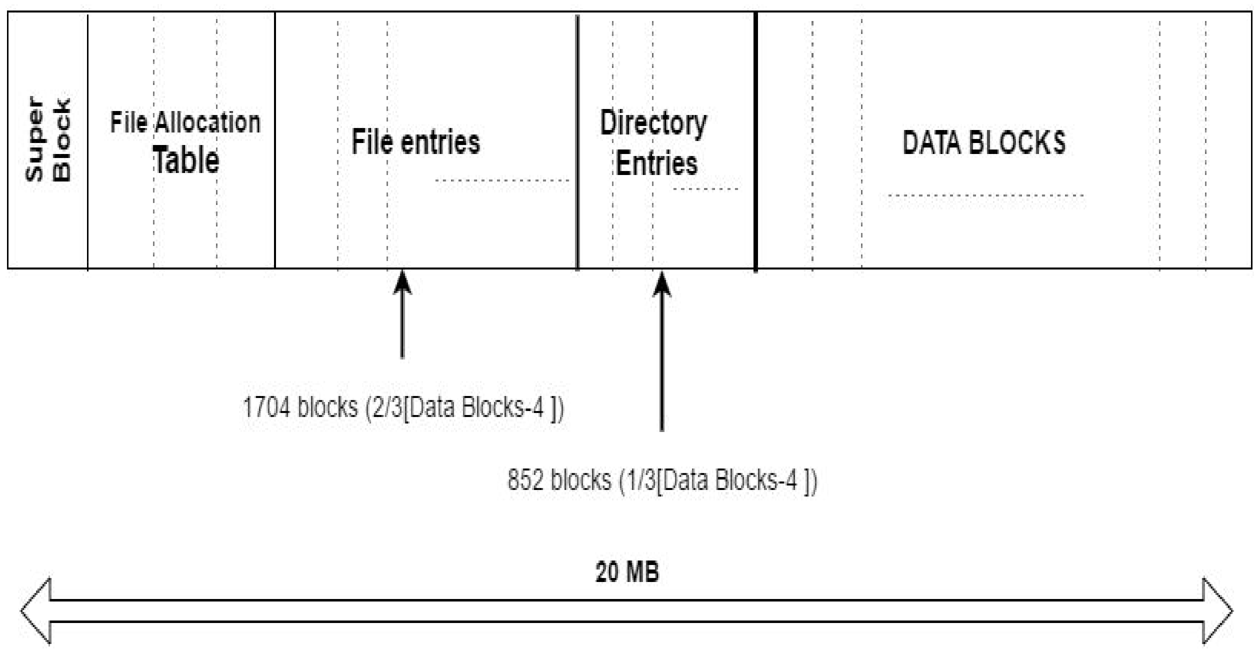
In version 2 , hierarchical directory system is applied that is shown in the below figure.



The working i.e the mount , unmount filesystem along with open and closing of disks remain same.Whereas all the file system functions will change accordingly.

The structure of the disk will be modified according to hierarchical directory implementation and can be seen in the figure below:

Directory: Hierarchical Access, Version:2



In this version, the hierarchical implementation is carried out by creating a different structure 'directory' and also modifying the directory_entry structure defined earlier.The directory are seen as a tree structure in which root is the bin or root directory and each directory will have its own sub- directory , there may be directories which will have only file entries in them.

Better Search in Version_2:

When a command create_dir(resemblance to mkdir) for new directory is given, the filename as well as the path of new file's position is also mentioned.The path is passed as an input to the function so that we can compare the path's elements to the tree elements one by one in an efficient manner(similar to search algorithm of tree data structure).Ex. Path given is A/B/C , implies at directory C create new directory with a given name,We know '/ or bin' is starting directory and according to the problem A must be its sub directory so a search operation will be done(linear search) and after finding directory A, will search in the subdirectory of A(children of A)and find B. Similarly find C in B, this search algorithm is more efficient as when I've selected the sub-dir of B by linear search I've avoided the search in rest directories (subdirectories of root except A) , hence his algorithm becomes more time saving than previous model where linear search was conducted and all children of root will be searched.

Similarly for all other cases of create and delete, as the version_2 is more efficient with better running time ,these cases are operated at a better speed in this version.

Delivered:

In version_1:

Proper functioning FAT file system on a virtual disk of 20MB with a sequential directory implementation and all functions listed above in a working condition.

In version_2:

Hierarchical Directory implementation with working functions for : directory_creation(create_dir) and optimised version of directory search (search).

FUTURE SCOPE:

As we have used hierarchical system and applied the algorithm consequently, there are various other approaches available for directory implementation such as considering directory as a B tree or an AVL tree. As all these algorithms have their own running time, tradeoffs and order of efficiency we can choose any of them depending upon our needs of the filesystem we are implementing.

APPLICATION:
The project enables us to better understand the working of various directory organization systems, understand the tradeoffs that are involved while selecting among the various algorithms and also able to devise an efficient algorithm on the basis of our understanding. Such an efficient algorithm will lead to a faster implementation of the directory system thus saving a lot of CPU time and in turn saving human capital.

CONCLUSION:

As the implementation is sequential access model, the time required for creating or searching a file/directory is equivalent to time required to perform linear search i.e of order O(n), n is number of entries in the FAT table.When we implement directories in form of a hierarchy and modify the create,search and delete algorithm of directory the time required for searching becomes logarithmic and hence is more efficient i.e. faster and time-cost effective . Therefore, directory implementation in a hierarchical methodology performs better than linear while searching.