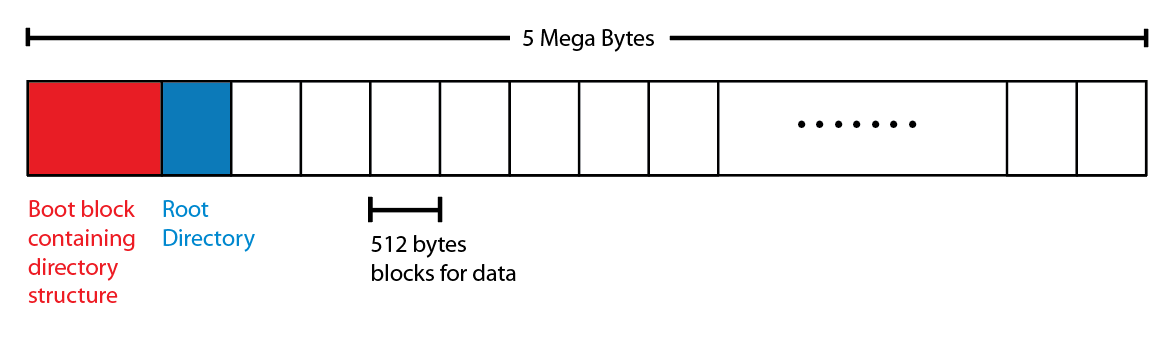
**Likhon Gomes (Leo)**

**LAB 4** File Systems

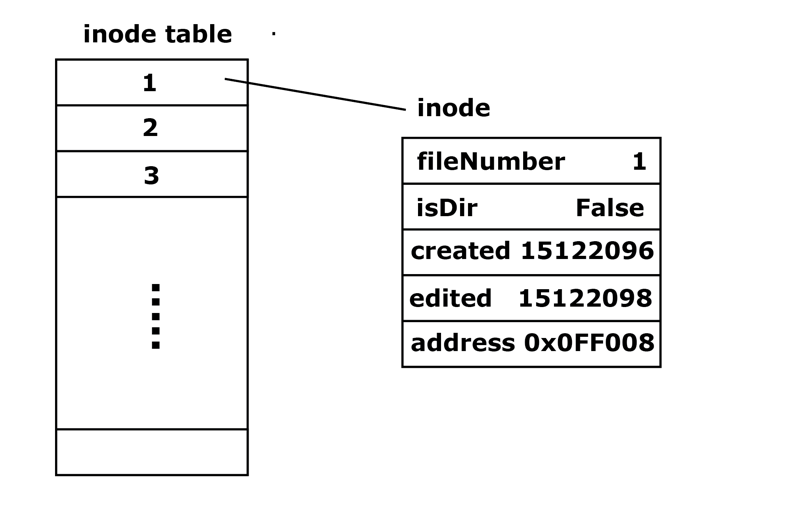
**Diagram of Virtual Disk Allocation**



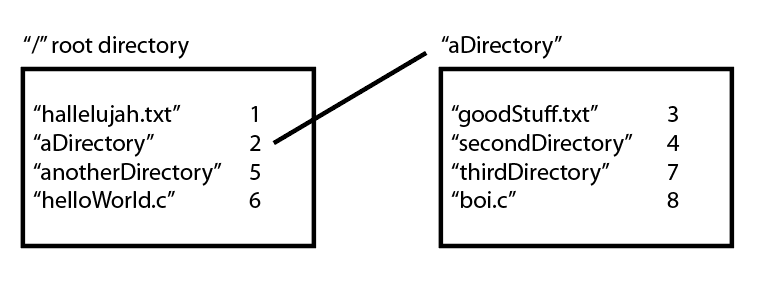
The boot block of the disk contains the data structure for the file system. It is created by format() function. It holds inode tables and inodes with metadata for all directories and files.

**Directory & File Structure**

Picture provided below structure of a file / directory



Each inode contains metadata about the file/directory. For example: The filename, a Boolean named isDir to specify if the file is a directory, two unix time stamps that specify the time when the file was created and when it was last modified (opened) and a final and the most important piece of information which is the actual address of the file in which it lives.



Each directories itself contain a list of file names and their inode numbers. For example in the picture above the root and aDirectory +directory is show along with some file/directories and their inode numbers.

**Functionalities**

Here are the functions listed below that I plan to implement to my filesystem.

• format()

* Takes a file pointer to an empty virtual disk as an argument
* Returns a new pointer to the formatted disk
* This function allocates a sector on the disk for metadata on files, and then divide rest of the disk for 512bytes chunks to store data.

• makeFile()

* Arguments
  + Address of the Directory where the file will live
  + Name of the file
* Return 0 on success and something else on failure
* This function creates the inode for the file and associates the name and required metadata with the inode and then add the inode to the inode table.

•openFile()

* Takes a directory path for where to look for the file and it’s name
* Returns a pointer to the virtual file object
* This function finds the file in the system if not found it calls makeFile() function to make a new file to make that has the same name.

•closeFile()

* Takes pointer to the virtual file to open the object.
* Returns 0 on success and anything else on failure.
* This function disconnects the file’s pointer from the inode table and turns it into a null pointer.

•Rm()

* Takes path of the file in the system and its name
* Returns 0 on success, otherwise on failure.
* Removes the name & the number of the file from the directory & removes the inode from the inode table.

•makeDir()

* Same as makefile but in this function the variable isDir is set to true.

•Write()

* Pointer to the object’s location where the file will be written, the data that’s going to be written in the file.
* Returns 0 on success or otherwise on failure.
* This function uses fseek() to navigate to the current location within the file, then uses fread to read the data.

•getFileNum()

* Takes the address of the path where the file lays and the name of the file.
* Returns the int value of the file’s number or -1 if not found.
* This function iterates through the directory and looks for the file.

•addInode()

* Takes the inode object
* Is a void function

readInode()

* Takes the file number to be read.
* Returns the iNode object or else null if it doesn’t exist.
* This function iterates through the inode table, scans for the file. If the file is found it reads through the file and returns it.

**Pseudocode**

Include <stdbool.h>

Include <stdio.h>

Include <sys/mman.h>

Include <time.h>

Define MAX\_SIZE 4096

Define BLOCK\_SIZE 512

Define MIN\_FILENAME 8

Define MAX\_FILENAME 64

//Global Variables

File \*filesystem // this is the file that will simulate the hard disk in the filesystem.

Inodetable \*iTable //pointer to the inode table

Int nextFileNum // increments each time a new file is created to give it the next sequential number

//Classes

Class virtFile{

Int current // the current location

Int end // the end of the file

}

Class data{

Int filenum

Char \*file

}

Class inode{

Int fileNum

Bool isDir

Time\_t created

Time\_t edited

}

Class inodeTable {

Inode \*first // address of the first node

Inode \*last //address of the last inode

Int length

}

//main function

Int main(){

Filesystem = fopen(“filesystem”,”w+”);

Format(myfs);

Char \*dirName = “newDirectory”;

Char \* filename = newFile.txt

char\*​ ​data​ ​=​ ​“The​ ​quick​ ​brown​ ​fox​ ​jumps​ ​over​ ​the​ ​lazy​ ​dog.\n”;

mkdir(dirName);  
mkfile(dirName,​ ​fileName);  
virtfile\*​ ​vf;  
Vf​ ​=​ openfile(dirName,​ ​fileName);  
write(vf,​ ​data);  
close(vf);

Virtualfile\*​ ​vf\_reopen;  
vf\_reopen​ ​=​ ​my\_openfile(dirname,​ ​filename);

char\*​ ​readdata;

my\_read(reopen,​ ​readdata,​ ​46);

if(strcmp(readdata,​ ​data))​ ​==​ ​match)

Print​ ​“Data​ ​successfully​ ​written​ ​to​ ​&​ ​read​ ​from​ ​file.”

Else

Print​ ​“Data​ ​was​ ​not​ ​successfully​ ​written​ ​to​ ​&​ ​read​ ​from​ ​file.”

Return​ ​0;

}

//​ ​format​ ​the​ ​disk  
  
Void​ ​​**format**​(FILE\*​ filesystem)​ ​{

//memory​ ​map​ ​the​ ​passed​ ​in​ ​pointer mmap(filesystem);

//initialize​ ​root​ ​directory mkdir(“”,​ ​“root”);

//initialize​ ​iTable

iTable->first​ ​=​ ​root;

}

int​ ​​**mkfile**​(char\*​ ​path,​ ​char\*​ ​fileName)​ ​{

if(sizeof(filename)​ ​<​ ​MIN\_FILENAME) Print​ ​“File​ ​name​ ​invalid:​ ​too​ ​short.” Return​ ​1;

Else​ ​if(sizeof(filename)​ ​>​ ​MAX\_FILENAME) Print​ ​“File​ ​name​ ​invalid:​ ​too​ ​long.”; Return​ ​1;

//create​ ​inode  
Inode​ ​file;  
time\_t​ ​curtime;  
curtime=​ ​time(NULL);  
file->created​ ​=​ ​curtime;  
file->edited​ ​=​ ​curtime;  
file->isDir​ ​=​ ​FALSE;  
file->filenum​ ​=​ ​++nextfilenumber;​ ​//global​ ​var

//add​ ​to​ ​inode​ ​table addItable(file);

Return​ ​0;

}

// ​open​ ​file  
//Struct​ ​for​ ​current​ ​location​ ​in​ ​the​ ​file​ ​and​ ​what​ ​file​ ​it​ ​is  
virtualfile\*​ ​​**my\_openfile**​(char\*​ ​path,​ ​char\*​ ​filename)​ ​{

if(sizeof(fileName)​ ​<​ ​FILENAME\_MIN\_SIZE)

Print​ ​“File​ ​name​ ​invalid:​ ​too​ ​short.”

Return​ ​NULL;

Else​ ​if(sizeof(filename)​ ​>​ ​FILENAME\_MAX\_SIZE)

Print​ ​“File​ ​name​ ​invalid:​ ​too​ ​long.”;

Return​ ​NULL;

//check​ ​if​ ​file​ ​exists​ ​by​ ​finding​ ​it​ ​in​ ​the​ ​directory fseek(filesystem,​ ​​*location*​ ​*of*​ ​*directory*​);  
Int​ ​fileNum​ ​=​ ​getFileNum(fileName);

//find​ ​its​ ​inode  
Inode​ ​thisinode​ ​=​ ​readInode (fileNum); if(thisinode​ ​==​ ​NULL)

Print​ ​“file​ ​not​ ​found.​ ​Creating​ ​%filename​ ​at​ ​%path.”

mkfile(path,​ ​fileName); if(thisinode->isDir​ ​==​ ​TRUE)

Print​ ​“Can’t​ ​open:​ ​this​ ​is​ ​a​ ​directory​ ​not​ ​a​ ​file.”

Return​ ​NULL; thisinode->edited​ ​=​ ​curtime; time\_t​ ​curtime;  
curtime=​ ​time(NULL);

virtfile​ ​\*vf;  
vf->current​ ​=​ ​address​ ​of​ ​file;  
vf->end​ ​=​ ​address​ ​of​ ​file​ ​+​ ​size​ ​of​ ​file; Return​ ​vf;

}

int​ ​​**closefile**​(virtfile​ ​\*vf)​ ​{

vf​ ​=​ ​NULL;

if(vf​ ​==​ ​NULL)

Print​ ​“cannot​ ​close​ ​file;​ ​file​ ​is​ ​not​ ​open.”

Return​ ​1;

Else

f​ ​=​ ​NULL;

Return​ ​0;

}

//Make directory

int​ ​​**mkdir**​(char\*​ ​path,​ ​char\*​ ​dirname)​ ​{

if(sizeof(dirname)​ ​<​ ​FILENAME\_MIN\_SIZE)

Print​ ​“Directory​ ​name​ ​invalid:​ ​too​ ​short.”

Return​ ​1;  
Else​ ​if(sizeof(dirname)​ ​>​ ​FILENAME\_MAX\_SIZE)

Print​ ​“Directory​ ​name​ ​invalid:​ ​too​ ​long.”;

Return​ ​1; Inode​ ​dir;

time\_t​ ​curtime;  
curtime=​ ​time(NULL);  
dir->created​ ​=​ ​curtime;  
dir->edited​ ​=​ ​curtime;  
dir->isDir​ ​=​ ​TRUE;  
dir->filenum​ ​=​ ​++nextfilenumber;​

Return​ ​0;

}

// delete a file

Int​ ​​**Rm**​(char\*​ ​path,​ ​char\*​ ​name)​ ​{

Int​ ​filenum​ ​=​ ​getFileNum(char\*​ ​filename);  
*Navigate*​ ​*to*​ ​*the*​ ​*directory*​ ​*and*​ ​*remove*​ ​*its*​ ​*name,number*​ ​*pair Find*​ ​*this*​ ​*file*​ ​*in*​ ​*the*​ ​iTable​ ​*remove*​ ​*its*​ ​*inode*​ ​*entry*

Return​ ​0;

}

//write a file

Int​ ​​W**rite**​(virtualfile\*​ ​vf,​ ​char\*​ ​data)​ ​{

fseek(myfs,​ ​vf->current);

fwrite(data,​ ​myfs);

Return​ ​0;

}

Int​ ​​R**ead**​(virtualfile\*​ ​f,​ ​char\*​ ​buf,​ ​int​ ​size)​ ​{

fseek(filesystem,​ ​vf->current);

fread(data,​ ​size,​ ​filesystem);

Return​ ​0;

}

int​ ​​**getFileNum**​(char\*​ ​path,​ ​char\*​ ​filename)​ ​{

For​ ​each​ ​data​ ​in​ ​thisdirectory

If​ data->name​ ​=​ ​fileName

Return​ ​data->number;

Return​ ​-1;

}

Void​ ​​addItable(inode​ ​file)​ ​{

fseek(myfs,​ ​​*location*​ ​*of*​ ​*iTable*);

fwrite(file,​ ​myfs);

}

inode​ ​​**readInode**​(int​ ​fileNum)​ ​{

fseek(filesystem,​ ​​*location*​ ​*of*​ ​*inode*​ ​*table*​);

char\*​ ​buf;  
while(not​ ​at​ ​end​ ​of​ ​table)

fscanf(buf,​ ​filesystem);

if(buf​ ​==​ ​fileNum) break;

Inode​ ​i; i->fileNum​ ​=​ ​buf; fscanf(buf,​ ​filesystem);

i->isDir​ ​=​ ​buf;

fscanf(buf,​ ​filesystem);

i->created​ ​=​ ​buf;

fscanf(buf,​ ​filesystem);

i->edited​ ​=​ ​buf;

}