Unix File System

Unix File System

Objective:

Using the C++ language Implement a inode bade Unix File System, the file system has the following functionalities, it has a pre-allocated 16MB in memory ad the store for the file system, this space is divided into 1KB blocks, each inode has 64 bytes of data inside the inode there is store 2 bytes for the file size, 3 sets of 8 bytes for the access, modification and creation time, a 14 byte char array for the owner 1 byte for the protection and 1 byte for the number of inodes pointing to this inode, 10 direct address blocks that are 2 bytes each pointing to blocks of data and one indirect block pointing to a block full of pointers to blocks of data.

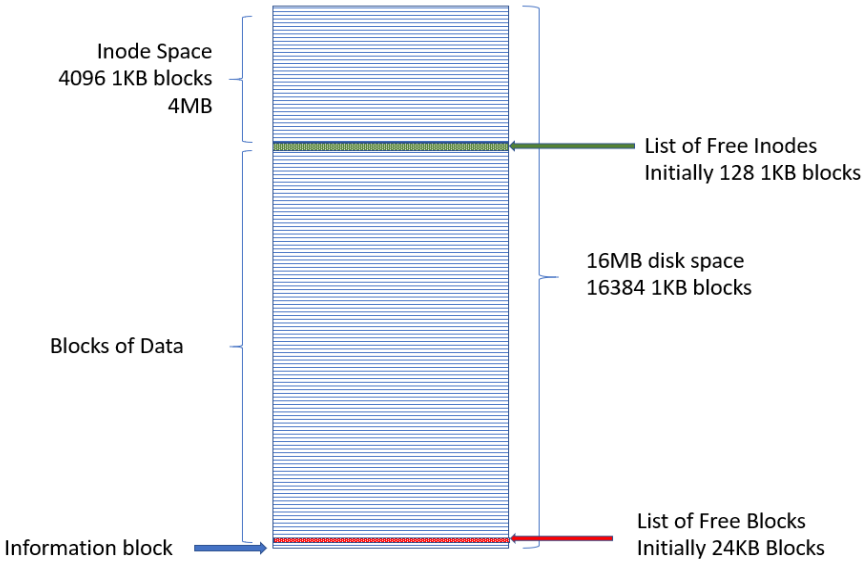
The first few blocks store the inode with the very first inode pointing to the root directory, random strings are written into the text files with a given number for the size of the file in bites.

The following commands are use in the project:

- createFile *filename fileSize* which will write a file to the current directory with the size given in bytes.
- deleteFile *filename* which is used to delete a file with a given name in the current directory.
- readFile will read the files inode and the random strings stored in the file.
- createDir *dirName* this command will create a directory in the current directory.
- readDir will read the current directory inode and the inode address stores in the directory
- moveDir *dirName* will move into a given directory in the currentdirectory or if given for example moveDir Robert/documents/SCUT will move to the directory which is on the currentDirectory then will move to documents which is in the Robert directory and finally to the directory SCUT which is in the documents directory for easier movement in the tree.
- deleteDir which will delete the current directory and put you in the deleted directories parent.
- copyFile *sourceFile destinationFile* this function will copy one file to another.
- infoBlock this will shoe the contents of the information block which keeps vital information about the disk.

Usage of Memory:

The first thing I want to talk about is the memory which from here on out I will refer to it as disk since that is what I called it in the code and I got used to referring it to disk. The 16MB disk space has two main areas the Inode Space and the Data Blocks, The Inode Space is from block 0 – 4095 making it a total of 4,194,304 bytes or 4MB for simplicity, therefor taking a quarter of the disk space, the reason for such a massive size is that each inode is 64 bytes and an inode address is two bytes long therefore the total number of inodes is 65,536, to store that many inodes you will need (65,536 * 64) 4MB of space.



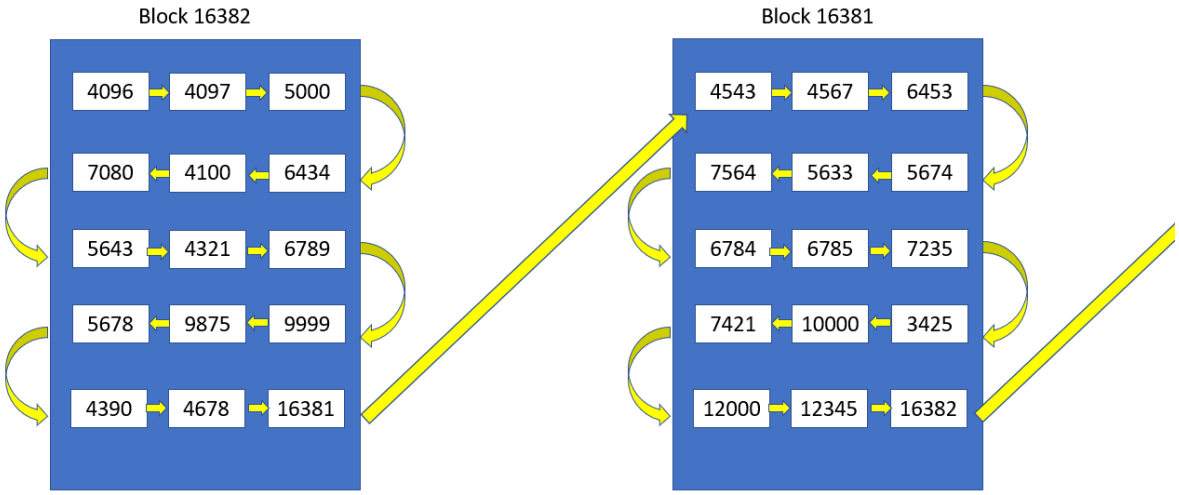
The inodes are stores at the top of the disk space is for easy hashing into a require area for example lets say that we want to access Inode 85 then we first figure out which block is in by figuring out how many bytes 85 inodes would fill so 85*64 = 5440 bytes and if we divided that by the block size which is 1kb we get (5440/1024) 5 and the remainder would be 320 so therefore is means that inode 85 is stored from 320-383 in block 5 in disk.

The Blocks of Data area in disk is where you store everything else being the text data, directories stored in an Inode the list of free blocks, the list of free inodes or the information block

Free Lists

The first thing to mention is the free list which stores all blocks that are not being used to store data, when the disk is being made the list will start at the second to last block the reason for this is that the last block in the disk is used for the storage of the information block, once the list is started it will append to itself all blocks starting from the first block not in the inode space (block 4096) and will stop once it has reached itself on the list since the blocks after that have been taken up by the free list and the information block.

When an block has been removed from the list the list will move to the next block and put a 0 where there once was a reference to a free list block, the last block of a free list block will not have a free block, instead it will have a pointer to the next block of the free block list and then add the now empty free list block to the top of the list, therefore while the free list takes quite some space by the time the disk is near full the free list will be but a fraction of its original size. Whenever an inode has been removed from an inode or the inode list has emptied this inode will append to the top of the free list ready for use next time is needed.



The list of free inodes is essentially the same as the inode list with the biggest differences being that the inode list does not have free block address but instead has address for free inode address in memory therefore the list will go from 1- 65535 the reason that it does not start at 0 is because that is taken by root which is created when the disk is created, the other big different is that the inode list starts at the begging of Blocks of Data space because it is created right after the free list is created therefore taking the first blocks of free space from the list. It is possible to finish all free blocks before finishing inodes since if we fill up the disk with empty directories, each directory will need to take a free block to store its parent and itself, this way disk will be filled with used 1kb blocks rather quickly but all this block will be using a very small percentage of the 1kb blocks.

The free block list and the inode list are made from a double linked list structured for easy access to any of the items within the list.

The information block is an especial block that is stored at the end of the disk space, this block stores the number of free blocks and inode available and the first block of the list of free blocks and the first block of the list of free inodes.

Structure of Objects:

Inode Structure

The inode structure consist of 64 bytes that contain the following:

- 2 bytes to store the size of the file
- 8 bytes to store the time created;
- 8 bytes to store the time last accessed
- 8 bytes to store the time last modified
- 14-byte char that store the owner of the inode
- 1 byte to store the protection
- 1 byte to store the count of inodes pointing to this inode
- 10 2 bytes address that point to a block of data
- 1 2 bytes indirect address that points to a block full of pointers to blocks of data

Giving a total of 64 bytes each inode can store up to 534,528 bytes of data, every time an inode is accessed the accessed time is changed to the current time, if the inode is changed in anyway the modified time will be changed to the current time, the created time will stay the same, each directory is its own owner however if a text file is created inside the directory the owner will be the directory that the file is in, the protection byte will tell what is allowed to be done to

this directory where is just reading or reading and writing, the count byte will keep track of the number of inodes pointing to this inode, lastly and probably the most important is the address blocks, the first 10 address blocks will contain a pointer to a block of data somewhere in disk full of the inodes data, the last address is an indirect address, this address points to a block that is full of address to blocks somewhere in disk filled with the inodes data, the address will be filled as needed by the inode.

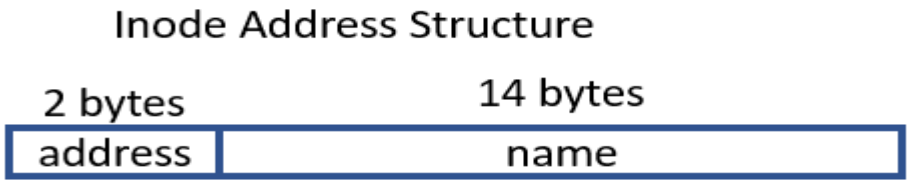
Inode Structure

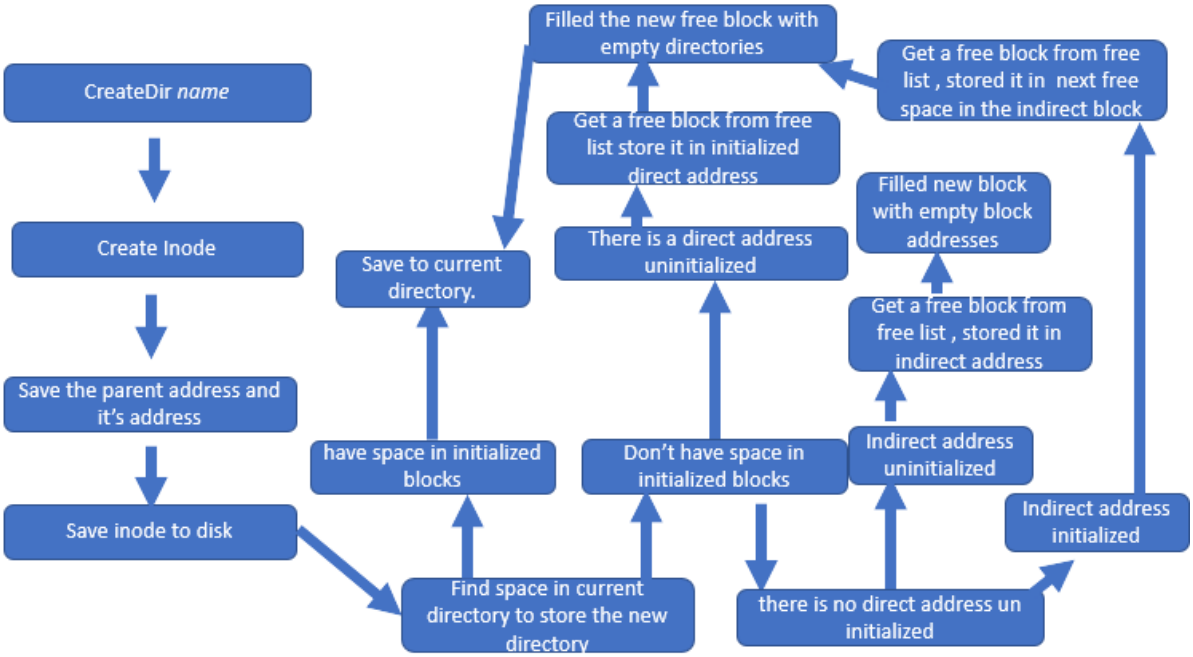
bytes	content
0-1	File size
2-9	Creation time
10-17	Access time
18-25	Modification time
26-39	owner
40-41	protection
42-43	count
44-45	Direct address 0
44-45	Direct address 1
44-45	Direct address 2
44-45	Direct address 3
44-45	Direct address 4
44-45	Direct address 5
44-45	Direct address 6
44-45	Direct address 7
44-45	Direct address 8
44-45	Direct address 9
44-45	Indirect address

Inode Address Structure

An inode address structure is made from a 16 bytes field that contains 2 things a 2 bytes address that points somewhere in the inode space and a 14 characters array to write the name of the file or directory this address will be appended into a directories data blocks and will take as many free blocks and inodes as needed from the free lists. The maximum amount of inode address structures that can be stored in a single directory is 64 and in theory that maxim amount

of inode address that can be stores in a singles directory is (522*64)33408 but for every inode it needs at least one free block so all free block will be depleted long before we can fill a single directory.



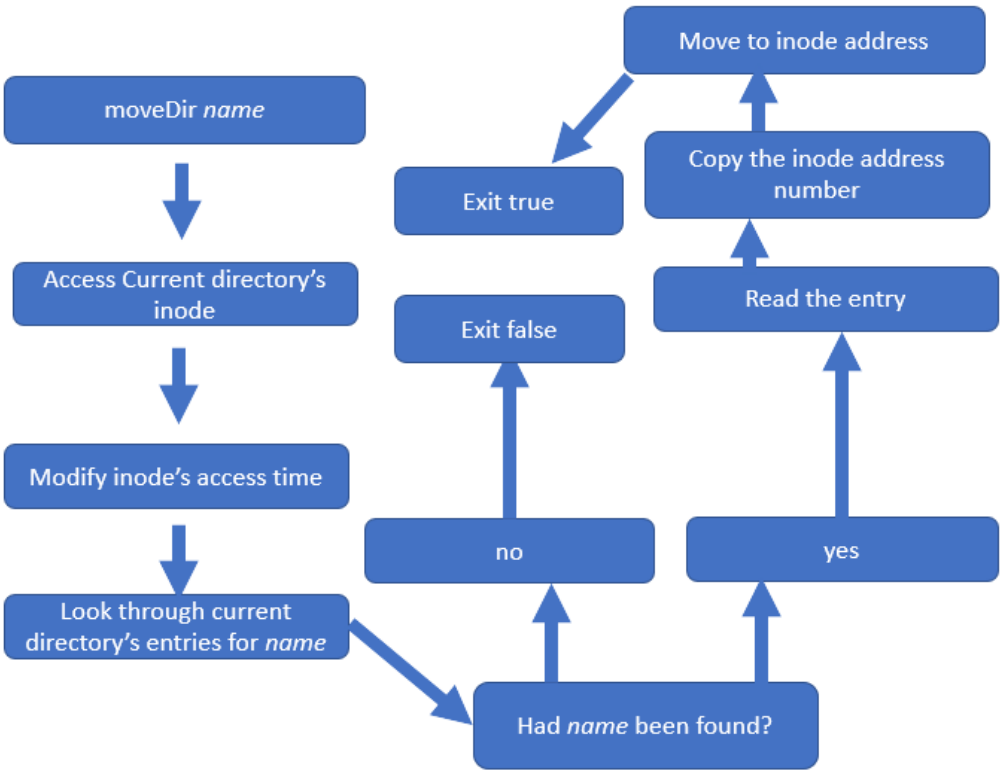


Directory Creation:

Upon creation of a directory an inode will be taken from the inode free list and an inode structure will be created with the access modification and creation time showing the time now, it will take a single free block which will be stored in the inode's indirect_address0 to store 2 directory entries "." Which will point to itself and ".." that will point to its parent then it will store this inode in disk using the inode address given,

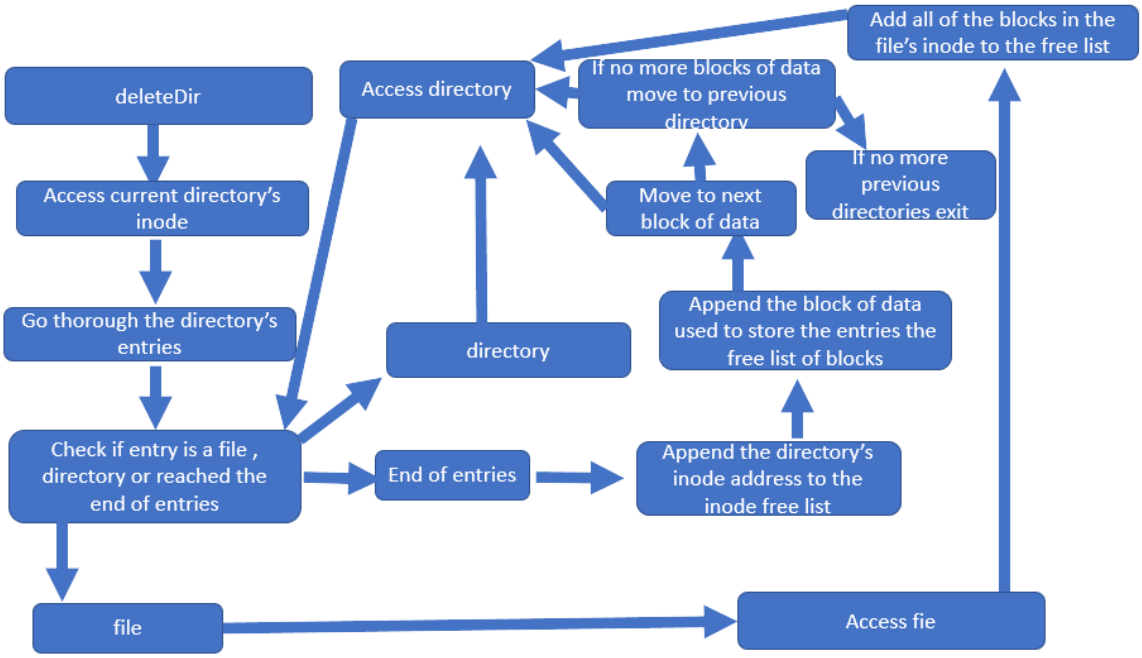
Next it will append the directory's inode address and name to the parent directory, which would be the current one, if there is still space in one of the initialized blocks then it will store the new directory there. if all initialized direct address blocks in the list are full but there is still one uninitialized direct address then a new free block will be obtained from the free list ,

we will fill this block with empty directories and appended the directory created to this block, if to append the new directory we have to access the indirect address then it will be treated as a especial case and two of the following things can happen, if the indirect address has not been initialized in this directory then a new free block is obtained from the free list, this block will be filled with empty block address and then stored in the inode's indirect address then a another free block is taken from the free list and filled with empty directories and it is this block where the new directory will be stored, if the indirect address has been initialized and all the blocks that have been initialized in the indirect block are full then a new free blocks will be taken from the free list and filled with empty directories and the new directory will be stored here.



Moving to directory:

Using the command moveDir *filename* the program will look in the current directory for any entry that has the same name given, the functions will load the current directory's inode into memory it will first modify the current directory's inode's access time then it will check through all initialized blocks of direct and indirect addresses for the given name if there is no entry with the name given then it will return false otherwise it will take the inode address number from the entry and move to that place in disk.



Delete Directory:

Using the deleteDir function will delete the current directory and all the directories within in it the first thing it will be do is find its parents inode address, save it to memory then it will go through every entry in the directory if it is a file entry, we only need to add all the blocks stored in the file's inode if it is directory we enter the directory access all its files and directories and delete things from them as needed once we come back to our current directory we add all direct and indirect addresses that are stored in the entries to the free list and add their inodes to the inode free list and lastly it will add is own inode address to the inode list and send all of its direct and indirect addresses to the free list.

Reading directory:

Using the readDir command will read the current directory, it will first access the inode modify its access time and the output all information about the inode, then it will go through every single initialized block in the inode and output the directories with their address first then next to it the entry's name.

MoveDir

```
moveDir robert/documents
readDir

file size       32
creation time   1525943607
last acessed    1525943698
last modified   1525943607
owner           documents
protection      0
count           1
address 0       4227
address 1       0
address 2       0
address 3       0
address 4       0
address 5       0
address 6       0
address 7       0
address 8       0
address 9       0
indirect        0
2       .
1       ..
```

CreateDir

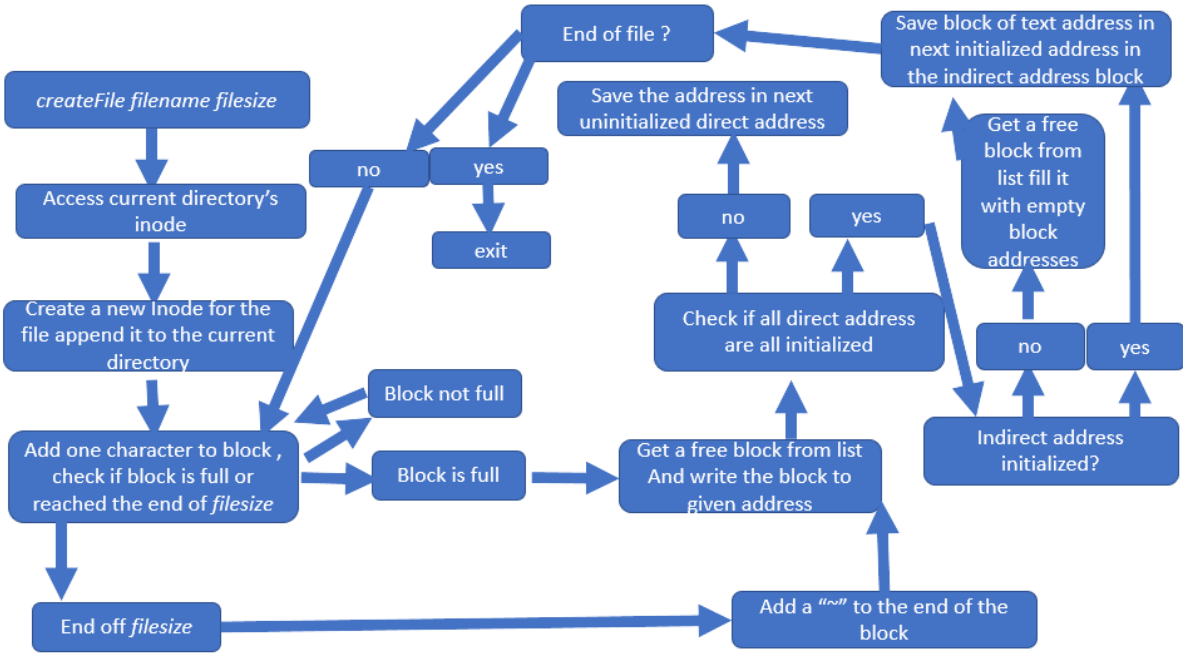
```
createDir projects
readDir

file size       48
creation time   1525943607
last acessed    1526019742
last modified   1526019742
owner           documents
protection      0
count           1
address 0       4227
address 1       0
address 2       0
address 3       0
address 4       0
address 5       0
address 6       0
address 7       0
address 8       0
address 9       0
indirect        0
2       .
1       ..
4       projects
```

ReadDir

```
moveDir projects
readDir

file size       32
creation time   1526019742
last acessed    1526019742
last modified   1526019742
owner           projects
protection      0
count           1
address 0       4249
address 1       0
address 2       0
address 3       0
address 4       0
address 5       0
address 6       0
address 7       0
address 8       0
address 9       0
indirect        0
4       .
2       ..
```

```
moveDir robert/documents/projects
readDir

file size       80
creation time   1526290221
last acessed    1526290257
last modified   1526290257
owner           projects
protection      0
count           1
address 0       4228
address 1       0
address 2       0
address 3       0
address 4       0
address 5       0
address 6       0
address 7       0
address 8       0
address 9       0
indirect        0
3       .
2       ..
4       junk
5       file1
6       file2
```

```
deleteDir
current file deleted<
readDir

file size       32
creation time   1526290187
last acessed    1526290315
last modified   1526290315
owner           documents
protection      0
count           1
address 0       4227
address 1       0
address 2       0
address 3       0
address 4       0
address 5       0
address 6       0
address 7       0
address 8       0
address 9       0
indirect        0
2       .
1       ..
```

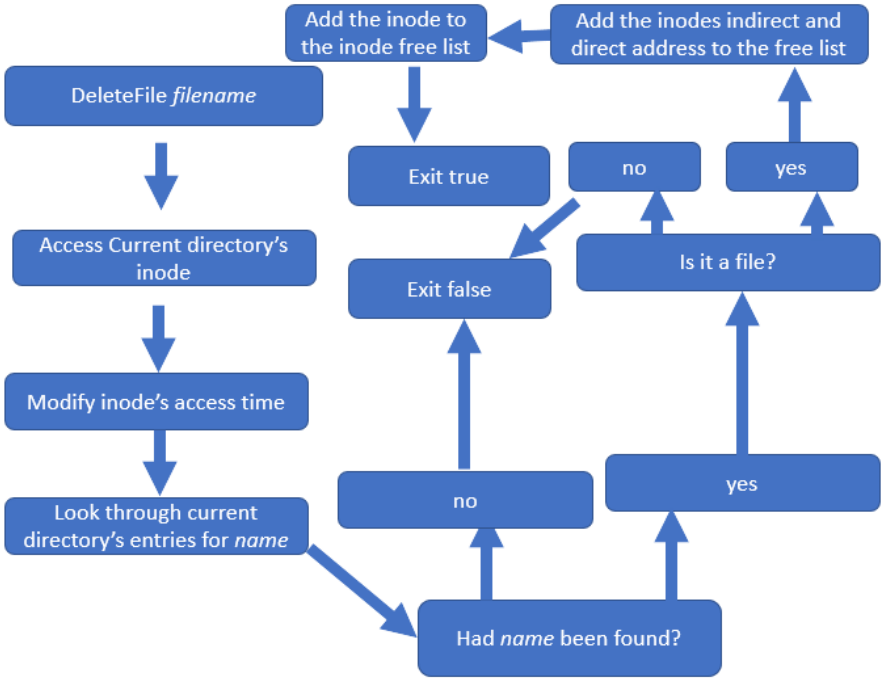


Create File:

The createFile *fileName fileSize* function will take a create a text file in the current directory, the size of this text file will be determined by the user who will input a file size in bytes(one byte is one character), first thing that happens is that an Inode is created for the file this inode will be similar to the directory inode except for two things the owner of the file will be the current directory the file is on and the protection will be of 1 meaning that this is a read only file, you may copy a file to this file but this will require for the data stored in this file to be deleted.

The amount of data blocks used by this file will be determined by the file size, at file creation a single character (from 32 -125 in the ASCII table)at a time will be added to a temporary block of data, when this block gets full with 1024 characters(bytes) then the content of this block will be copied to a data block in memory a free block from list will be taken and the block will be saved on that block address and the block address will be added to the inode, if we reach the indirect address before reaching an end to the file size a free block will be taken form the list and this free block will be filled with empty block addresses, then we grab another free block from list and append it to the beginning of the indirect block then write the block to the given address on disk and from then on when we want to write a block to disk from this will the block address will be appended to the indirect address block, once we reach the end of the file size a '~' will be added to the end of the block to indicate the end of the text file and

a free block from the list will be taken, the block will be saved to disk and the addressed saved to the inode.



Delete File:

The deleteFile *fileName* will delete a file in the current directory with the given name, the function will grab the inode from disk then go through every single block address that has been initialized in the file's inode and appended the block address to the free list of blocks, if the indirect address is reached then we load the indirect address to memory and appended all block addresses initialized the free list of blocks once we reached the end of initialized blocks in the indirect address then the indirect address block address is added to the free list of blocks, lastly the file's inode is appended to the free list of inodes and the entry deleted from the current directory.

Read File:

The readFile *fileName* will read a file in the current directory with the given name, it will first read it's inode information and then read all every single data block till a "~" is found.

Conclusion:

This project shows some of the basic functions of inodes in the Unix File System, it shoes how it uses inodes to access data in the disk, it is able to create directories that have data blocks

pointing to different areas in the inode space, and this inodes if they are directories will have more data blocks pointing to different areas in the inode space, each inode has enough space to point to 522 blocks of data 10 being direct and the other 512 stored in the indirect address block.

We are able to see how files are stored in the inode space and how it uses the data blocks addresses stored in the inode to connect read the files data. And we are able to see how the free list of inodes and free list of blocks are used in an operating system to store data that is not being currently used and using this method we can see that overhead is virtually none existent since by the time we are about to fill the disk the lists will be a fraction of their original size.