Abraham Schultz
**Temple CIS 3207**
Lab 4

Virtual Disk Design Document

DESIGN OVERVIEW

This is my design for a virtual disk. This disk will reside on top of the existing linux file system within a single file.

I will use a FAT as the basis for my virtual file system.

1. The maximum file size will be 32 mb which will be all of the blocks. Each block is 4KB or 4096 bytes. There will be 16,384 blocks but only half of them need to be used for storing data files. This Means that we should have **64 MB total** for our virtual disk, half of that being allocated for storing files.
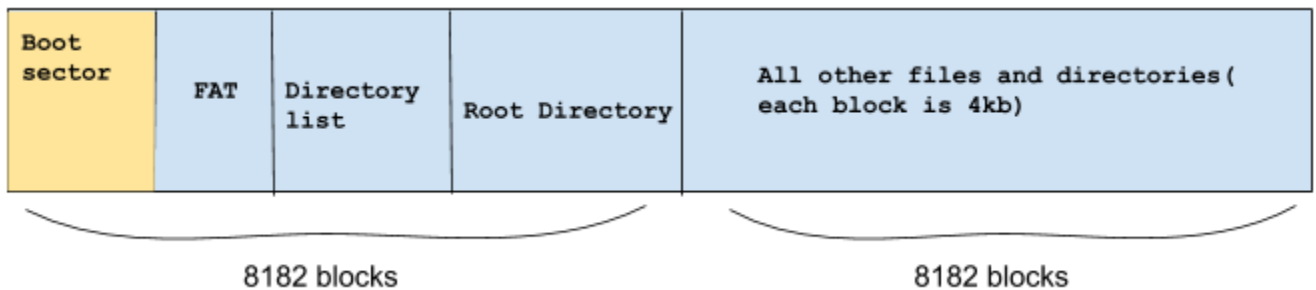
2. The FAT will act as one large central table to store information about the location of each new file we would like to add. It should be Essentially a list of linked lists. Where each linked list within is referencing the location of the blocks needed to store a particular file.

3. If we create a new file, first we will need to check if the size of that file is small enough to fit into one block. If so then we can assign just one block add add that block to a list on the FAT. Otherwise if the file requires more than one block we will need to break the file up into multiple blocks. The FAT will record these files as a linked list with the starting positions of each required block as an entry in that files linked list. In this way we can store a file in a non contiguous way taking advantage of gaps in the memory.

4. Every time a new file is created we will subtract one from the available disk count. We will also check for the current offset (that is the starting position in the disk memory to start writing to) we will then begin writing to the disk one block at a time assigning each block the starting position of the current offset making sure to check if the next block is used or not. If it is used then we assign the block and mark as used and update the current offset, otherwise we skip to the next block.

STRUCTS FOR PARTITIONS OF VIRTUAL DISK

The FAT Struct will look as follows it will be our main struct, which will keep track of files open and the blocks being used.

Struct FAT{

Int TotalBlocks = 16384

Int UnusedBlocks; // start at total blocks and subtract from as we go

Vector <File > FAT // a linked list containing linked lists of blocks.

} // struct FAT

The file struct will be a linked list of blocks

Struct File{

Vector <struct Block> blockList ; // this will contain the block holding the data

Int FileSize = blocklist.length * blockSize;

Int filePointer;   // pointer to current data byte

Int startingAddr; // address of first block used by this file

bool operator<(const startingAddr& rhs) const

{ // i think we can use this to compare file locations prioritizing lower address

return startingAddr> rhs.startingAddr;

}

} // end Struct File

The Block struct will be an element in the

Struct Block {

Char DATA [4000] // these file will only hold text for now. 4000 bytes = 4KB

Int addrNextBlock; // point to next block used -1 if not used

} // end Struct Block

// used to initialize super block

PSEUDO CODE AND FUNCTIONS

  // the main function will contain a central while loop that has a simple shell, i will   probably reuse some code from lab 2.

#declare constants and macros IE size of disk ect

> *Main() {*
>
>> *intitlize ();*
>>
>> *While (running == true) {*
>>
>>> *Char * buff // hold user input*
>>>
>>> *Getline ();*
>>>
>>> *Big switch case*
>>>
>>> *Switch ()*
>>>
>>>> *Case open()*
>>>>
>>>> *….*
>>>>
>>>> *Rest of the functions listed below will each be there own case*
>>>>
>>>> *Default*
>>>>
>>>> *Case unrecognized command try again*
>>>>
>>>> *}*
>>
>> *} // end while*

} // end main

> **int make_fs(char *disk_name); {**

MakeDisk(disk_name)

//create new FAT instance

Struct FAT = new FAT();

FAT.totalBlocks = TOTALBLOCKS

Struct FILE rootDir;   // create root directory on lowest memory blocks

} // end make_fs

int mount_fs(char *disk_name); {

// check if disk exists

If it exists

Open disk (disk_name)

Else

Error message

// we need to make the selected virtual disk available to edit and read and write from

} // end mount

int umount_fs(char *disk_name); {

Closedisk (disk_name)

} // end unmount

int fs_open(char *name);{

For (each item in FAT)

{

If FAT(i).name = name{

// open file

Return 1;

}

}

Return -1 // only here if no file found

}

int fs_close(int fildes);

exist or is not open, the function returns -1.

```
int fs_create(char *name){

    File tmpFile = File;

    File.name = name;

    //first check if we have space

    If (FAT unusedBlocks > 0) {

            FAT.insert(name)


    } // else we have no space

    printf("not enough space on disk \n")

            };
int fs_delete(char *name);{
//check if file exists
For (each item in FAT)

    {

    If FAT(i).name = name{

            FAT.erase(i)

            Return 1;

            }

    }

    Return -1 // only here if no file found


}
int fs_mkdir(char *name);{
}// end fs_mkdir
    int fs_read(int fildes, void *buf, size_t nbyte);{
// wrapper function for one provided by professor

}
    int fs_write(int fildes, void *buf, size_t nbyte);{
// wrapper function for one provided by professor

}
```

```
int fs_get_filesize(int fildes); {

    For (each element int FAT) {

        // check if file exists

        If (FAT(i)  = fildes ){

                Return FAT(i).fileSize;

        } // end if

    } // end for

} // end fs_get_filesize

    int fs_lseek(int fildes, off_t offset);{

} //end fs_lseek

    int fs_truncate(int fildes, off_t length);{

    //check if file exists

    // if file exists then check how big it its

    // if it is bigger than the length we must free the extra blocks

    // if file does not exist then return error message and -1

} //end fs_truncate
```

UNIT TESTS

We should be concerned generally with creating and editing files. Additionally, another important feature is persistence, that is this virtual disk should be able to be run, save files, then exit. And when i return they should still be there.

Below are some tests that I ran to verify that the program is fulfilling the project requirements:

1. Run shell and accept command to create filesystem

2. Accept command Mount file system

3. Create file

4. Accept command to Open file

5. Edit file
6. Delete file


```
  FILE DES: -1
  FILE START ADDR: 0
  FILE PARENT: 7913228


created new file file1
Please type a command
delete
Please type name of the new file to delete
file1
deleted file1 successfully
Please type a command
print


size of FAT 1

 used blocks : 16383
element in FAT: 0
   ***********************
  FILE NAME: /
  FILE SIZE: 32
  FILE DES: 0
  FILE START ADDR: 0
  FILE PARENT: 7913148


FILE BlOCK NUMBERS: 0


Please type a command
```

7. Exit


```
deleted file1 successfully
Please type a command
print


size of FAT 1

 used blocks : 16383
element in FAT: 0
   ***********************
  FILE NAME: /
  FILE SIZE: 32
  FILE DES: 0
  FILE START ADDR: 0
  FILE PARENT: 7913148


FILE BlOCK NUMBERS: 0


Please type a command
unmount
Please type name of the disk to unmount
testDisk
writing FAT to disk : 32 / 0 0 0 0  0

successfully wrote FAT to super block
disk closed correctly
Please type a command
exit
Goodbye
cis-lclient01:~/lab4/AbrahamSchultz 3207Labs/lab4/src>
```

8. Repeat steps 2 and 4 and attempt to access data persistently

```
cis-lclient01:~/lab4/AbrahamSchultz_3207Labs/lab4/src>hd testDisk
00000000  33 32 20 2f 20 30 20 30  20 30 20 30 20 20 30 20  |32 / 0 0 0  0 |
00000010  0a 00 00 00 00 00 00 00  01 00 00 00 20 6e 6f 6e  |............ non|
00000020  30 00 00 00 00 00 00 00  21 00 00 00 00 00 00 00  |0.......!.......|
00000030  74 65 73 74 44 69 73 6b  00 db 33 eb f8 7f 00 00  |testDisk..3.....|
00000040  20 52 48 2f 44 65 62 69  61 00 00 00 00 00 00 00  | RH/Debia.......|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  2f 00 00 00 00 00 00 00  |......../.......|
00000070  00 00 00 00 00 00 00 00  20 00 00 00 00 00 00 00  |........ .......|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  00 00 00 00 00 00 00 00  00 00 00 00 22 5c 65 5b  |............"\e[|
000000a0  48 22 3a 20 62 65 67 69  31 00 00 00 00 00 00 00  |H": begi1.......|
000000b0  60 bb 78 00 00 00 00 00  00 00 00 00 05 00 00 00  |`.x.............|
000000c0  00 00 00 00 00 00 00 00  01 00 00 00 00 24 65 6e  |.............$en|
000000d0  30 00 00 00 00 00 00 00  21 00 00 00 00 00 00 00  |0.......!.......|
000000e0  0d 00 78 00 00 00 00 00  00 00 00 00 00 00 00 00  |..x.............|
000000f0  00 00 00 00 00 00 00 00  21 00 00 00 00 00 00 00  |........!.......|
00000100  6e 65 77 00 f8 7f 00 00  88 db 33 eb f8 7f 00 00  |new.......3.....|
00000110  20 00 00 00 00 00 00 00  21 10 00 00 00 00 00 00  | .......!.......|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000004d0  01 00 00 00 00 00 00 00  10 36 78 00 00 00 00 00  |.........6x.....|
000004e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000900  00 00 00 00 00 00 00 00  e0 6d 90 eb f8 7f 00 00  |.........m......|
00000910  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
04000000
cis-lclient01:~/lab4/AbrahamSchultz_3207Labs/lab4/src>
```