Abraham Schultz
3207
Lab 4

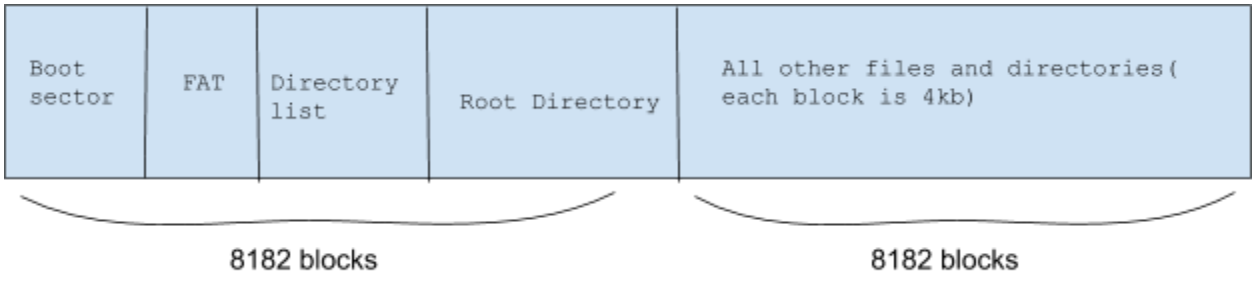
Virtual Disk Design Document

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 4000 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

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

} // struct FAT

The file struct will be a linked list of blocks

Struct File{

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

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

Struct BootSector{

Int totalBlocks // 16383

Int bytesPerBlock // 4000

String DiskName

StartingBlocks // blocks saved for initialization

} // end struct BootSector

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.

Main() {

intitlize ();

While (running == true) {

*Char * buff // hold user input*

Getline ();

Big switch case

Switch ()

Case open()

```
                    ….
                    Rest of the functions listed below
                    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
```

}

int mount_fs(char *disk_name);

int umount_fs(char *disk_name);

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(i).
                Return 1;
                }
        }
        Return -1 // only here if no file found

}
    int fs_mkdir(char *name);
    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);

int fs_lseek(int fildes, off_t offset);

int fs_truncate(int fildes, off_t length);

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.