MP7

Clarify: I finish the **basic requirement** and the **OPTION1**.

1. FileSystem Design

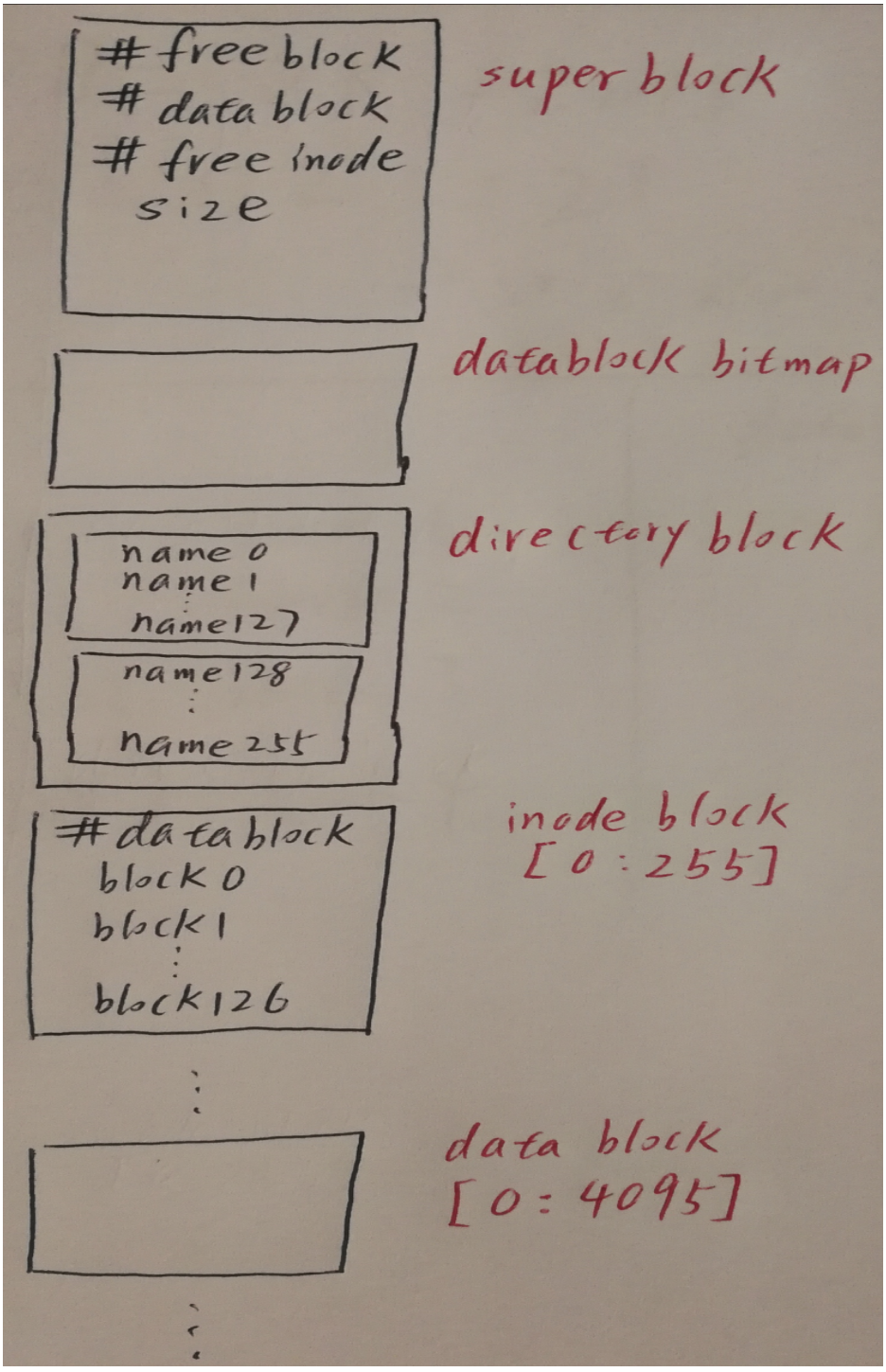


Figure1 Structure of FileSystem

As shown in Figure1, my FIleSystem has 1 SuperBlock(#freeblock, #datablock, #freeinode, size), 1 DataBlock Bitmap(512*8=4096 bits), 1 Directory Block(actually 2 blocks, 256 entries, which includes the filename indexing to the inode number), 256 inode blocks, and 4096 data blocks.

Class FileSystem

```
class FileSystem {

friend class File; /* -- not sure if we need this; feel free to delete */

private:
     /* -- DEFINE YOUR FILE SYSTEM DATA STRUCTURES HERE. */

    static int super_blockid; //{0};
    static int dbitmap_blockid; //{1};
    static int dir_blockid; //{2}; 2 blocks for dir_block
    static int inode_blockid; //{4};
    static int data_blockid; //{260};

    int freeblock_num, datablock_num, size, autoname, freeinode_num;

    SimpleDisk * disk;

    unsigned char super_blockbuf[BLOCKSIZE]; // BLOCKSIZE=512
    unsigned char dbitmap[BLOCKSIZE]; //BLOCKSIZE=512(B) => 2MB maxsize
    unsigned char dir_blockbuf[DBLOCKSIZE]; // DBLOCKSIZE=1024
    int dir_name[MAXFILENUM]; // MAXFILENUM=256, inode/file number max=256
```

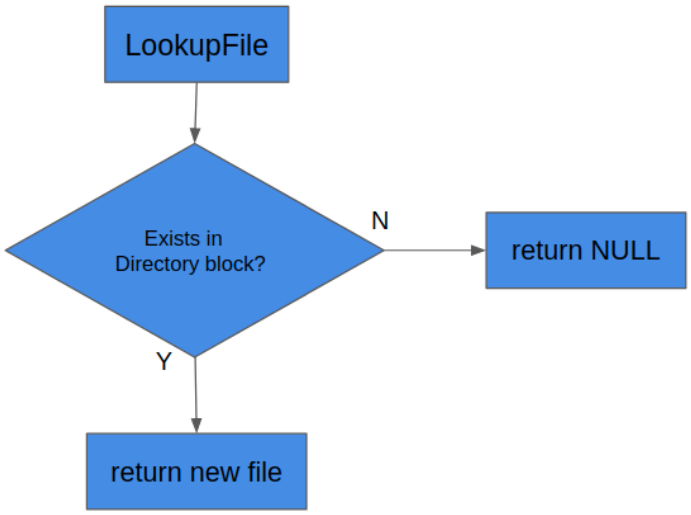
```
public:

    FileSystem();
    /* Just initializes local data structures. Does not connect to disk yet.
*/
    bool Mount(SimpleDisk * _disk);
    /* Associates this file system with a disk. Limit to at most one file sys
tem per disk.
     Returns true if operation successful (i.e. there is indeed a file system
 on the disk.) */

    static bool Format(SimpleDisk * _disk, unsigned int _size);
    /* Wipes any file system from the disk and installs an empty file syste
of given size. */

    File * LookupFile(int _file_id);
    /* Find file with given id in file system. If found, return the initializ
ed
     file object. Otherwise, return null. */
    bool CreateFile(int _file_id);
    /* Create file with given id in the file system. If file exists already,
     abort and return false. Otherwise, return true. */
    bool DeleteFile(int _file_id); /* Delete file with given id in the file s
ystem; free any disk block occupied by the file. */
```

FileSystem::**Mount**
(1)Load SuperBlock information from disk
(2)Load DataBitmap from disk
(3)Load Directory Block from disk

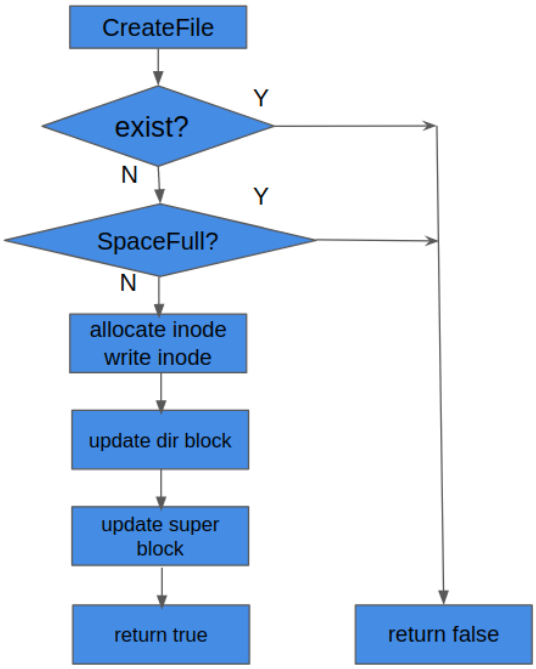
FileSystem::**Format**
(1)Write initial SuperBlock on disk
(2)Write empty(initial) DataBlock Bitmap on disk
(3)Write empty(initial) Directory Block on disk

(4)Write 256 empty(initial) inode blocks on disk

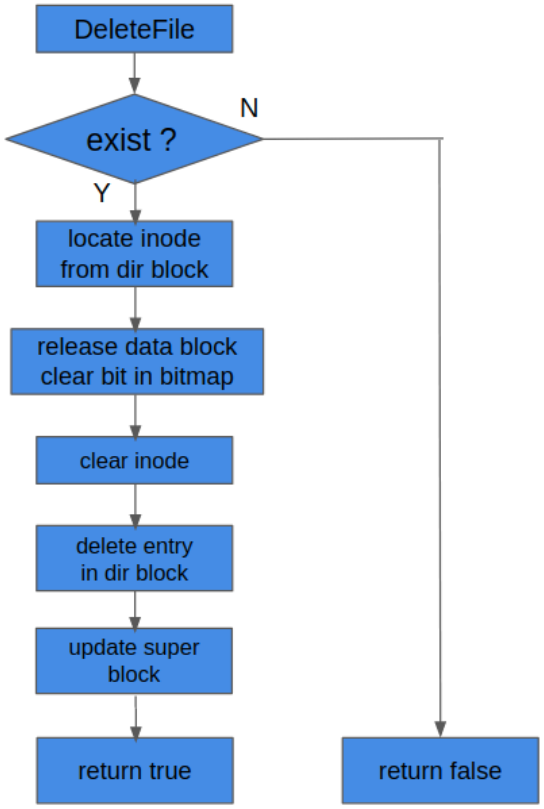
FileSystem::**LookupFile**



FileSystem::**CreateFile**



FileSystem::**DeleteFile**



Data Block Bitmap management

```
unsigned char FileSystem::getdbit(unsigned int index) {
    if (index >= datablock_num) {
        Console::puts("index=");Console::puti(index);
        Console::puts(" is larger than datablock_num=");
        Console::puti(datablock_num);Console::puts("\n");
        assert(false);
    }
    int nbyte = index >> 3;
    int offset = index & 0x7;
    int rshift = 7 - offset;
    return (dbitmap[nbyte]>>rshift) & 1;
}

void FileSystem::setdbit(unsigned int index) {
    if (index >= datablock_num) {
        Console::puts("index=");Console::puti(index);
        Console::puts(" is larger than datablock_num=");
        Console::puti(datablock_num);Console::puts("\n");
        assert(false);
    }
    int nbyte = index >> 3;
    int offset = index & 0x7;
    int lshift = 7 - offset;
    dbitmap[nbyte] |= (1<<lshift);
}
```

```
void FileSystem::cleardbit(unsigned int index) {
    if (index >= datablock_num) {
        Console::puts("index=");Console::puti(index);
        Console::puts(" is larger than datablock_num=");
        Console::puti(datablock_num);Console::puts("\n");
        assert(false);
    }
    int nbyte = index >> 3;
    int offset = index & 0x7;
    int rshift = 7 - offset;
    if ((dbitmap[nbyte]>>rshift) & 1)
        dbitmap[nbyte] ^= (0x80>>offset);
}

void FileSystem::sync_dbitmap() {
    disk->write(dbitmap_blockid, dbitmap);
}
```

Class **File**

```
class File  {

private:

    int pos, maxpos;
    int size;
    int datablock_num;
    int inode;

public:

    File(int _inode);

    int Read(unsigned int _n, char * _buf);

    void Write(unsigned int _n, const char * _buf);

    void Reset();

    void Rewrite();

    bool EoF();

    int GetPos();
    int GetSize();
    int GetDBN();
    int GetInodeNo();
```

File::**Read**

(1)Load inode block, according to inode

(2)According to the current file pos, to locate the data block index in the inode block, and thus further locate the data block. Read the data from the data block 1 byte by 1 byte into the buf, when this data block is finished reading, switch to read in the next data block, and so forth.

(3)read until to the end of this file or to the end of the buf.

File::**Write**

(1)First compute the total size for this write operation, if needed, allocate new blocks to this file. So we do not need to worry about the size limitation during the writing process.

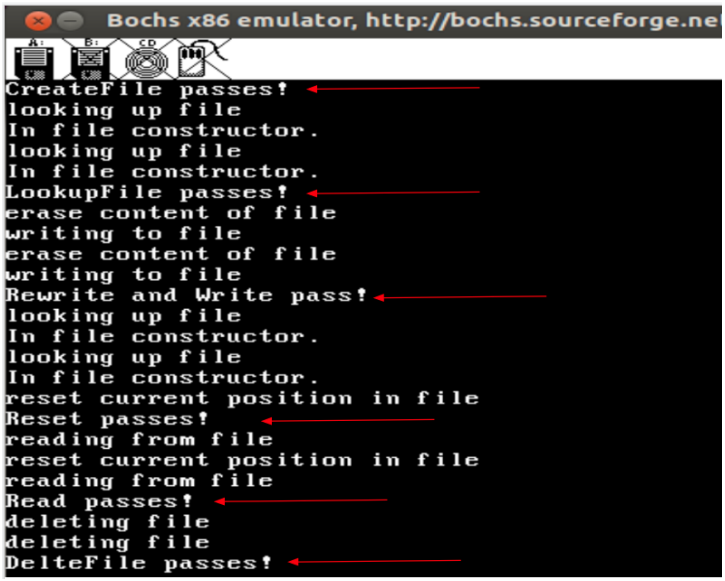
(2)Load inode block, according to inode

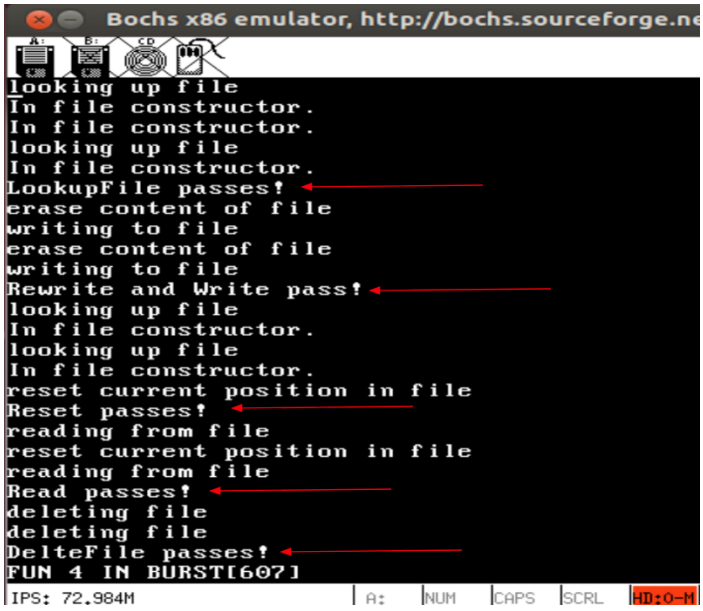
(3)According the current file pos, to locate the data block index in the inode block, and thus further locate the data block. Write the data from input buf into the corresponding data block 1 byte by 1 byte, when the current data block is written to be full, switch to the next data block and continue to write.

(4)write until all the characters in buf are written into the file, we finish.

Testing

As shown below, I successfully finish all the features, including CreateFile, LookupFile, Rewrite, Write, Reset, Read, and DeleteFile.





How to run my testing?
Inside my project directory, run the script run.sh.

```
guest@TA-virtualbox:~/workplace/MP7$ cat run.sh
#!/bin/bash

make clean -j$(nproc)
make -j$(nproc)

./copykernel.sh

bochs -q -f bochsrc.bxrc > myout
```

(OPTION1)Design a Thread-Safe FileSystem

Use **Read/Write Lock** mechanism.
Imply **Write Lock** to protect the critical section which writes on the FileSystem structures, like superblock, data block bitmap, directory block, inode block and data block.
Imply **Read Lock** to protect the critical section which only reads on the FileSystem structures.

(1)FileSystem
WriteLock protects CreateFile
WriteLock protects DeleteFile
ReadLock protects LookupFile

(2)File
WriteLock protects Write
WriteLock protects Rewrite
ReadLock protects Read
ReadLock protects Reset
ReadLock protects EoF