# Luke Selbeck, Greg Kitzmiller, Dallas Van Ess

# Bob Dimpsey

# March 16th, 2015

# CSS 430

## Assignment 5 Report

## Specification and Assumptions

We have implemented a single level file system along with eight system called outlined below. Luke and Greg were already a team and Dallas was unable to find a partner so we all teamed up.

## Assumptions

We assume that the space of a short is enough to index into the blocks.

### Sync

Copies all of the data in main memory to the disk. This includes the directory with all of the file names and lengths; other inodes; and the File Table. The directory is considered as inode 0.

### Format

Resets the disk and initializes the file system. You can specify a maximum number of files, and the superblock will create that many inodes, starting from block 1. Each inode is 32 bytes long, so 16 inodes can fit in a 512 byte block. e.g. If you format with a maximum file size of 65, 5 blocks will be allotted to inodes, from block 1 to block 5. After the inode blocks are the free blocks, which are available to be written in as file data. All of these blocks are initialized into a massive free list of all blocks after the superblock and the inode blocks. The way this list works is by putting the number of the next block in the list at the first four bytes of the current block.

After all blocks are initialized, the directory and the file table are created, which are the data structures that control the file, its attributes, and its functionality.

### Write

Takes a file table entry and a byte buffer as arguments. To be written to the file at the position indicated by the seek pointer in the file table entry. We check for errors such as space requirements, null references, or of the pointer goes beyond the end of the file. The superblock has a ‘stack’ of free blocks allowing for writing beyond already allocated filesize. If the seek pointer at a position where a file already has information beyond it, the file will be overwritten with new information. The Inode is updated with the changed metadata.

### Read

The read method takes a buffer and a file table entry and reads into the buffer. It has error checking regarding overflowing the buffer or reading past the length of the file (given a buffer larger than the file size). It reads block by block and finds new blocks based on Inodes ‘findTargetBlock’ method which takes the seek pointer and returns the target block. It uses array copy to copy into the return buffer. When it reaches either the end of the buffer or the end of the file it’ll return the buffer.

### Seek

The seek method changes the pointer in a file. You have the option to either just assign the pointer to a location (offset), or add/subtract from the current pointer location, or put the pointer at a specified distance from the end of the file. If the pointer at any point would go out of bounds, then it is just clamped to the edge of the file that it overflowed (either 0 or eof).

### Close

Close checks to see whether the delete flag is set and no other processes are using the file and if so, it’ll call the superblocks dealloc method. Otherwise, it will remove this processes entry from the file table. The superbloc removes the block and changes the location of the head.

### Delete

Deletes a file

### Fsize

Returns the size of a file

### Summary

All of these methods allow the user to create a file system which can manage files and save them systematically to a disk, or any form of hard memory.

## Description Of Components

### File System

The file system class contains all of the other classes. Instantiating it in memory instantiates a *file system* in memory. The methods and were defined in the slide deck and many are synchronized or have synchronized components due to accessing disk directly or metadata related to a file at a disk location. The file system has a directory which is just root ‘/’ with a bunch of files in it and the directory is basically just a file that is read into memory on instantiation of the file system.

### SuperBlock

Superblock is block zero and contains metadata about the filesystem such as: total number of blocks, total number of inodes, and the number of the first block that is unused.

### File Table Entry

The file table holds entries which contain metadata about open files and reference to their Inodes. This includes a count of the threads using the file and the mode that it’s opened in.

### Inode

The Inode has direct metadata about a file including its array of 11 direct pointers (indexes) and an indirect pointer after those. It has a flag for validity and has a count of the amount of file table entries which point to it and also contains the size of the file in bytes.

### TCB

Each user has a TCB which they own which uses its own filetable to manage files which you can read, write, open, and close. Meaning, each user has files that they can open and close independent of each other. This was already provided for us.

### Directory

Directory has a 1-d array of file sizes and a 2-d array of names where the index maps to the corresponding inode. The directory is the map that looks up the corresponding inode of a file based on its name. This is basically a file and is saved in the first inode used.

**Performance Functionality**

## Output

Script started on Wed 18 Mar 2015 07:33:01 PM PDT  
lps1994@uw1-320-11:~/CSS430/FileSystemProject$ java Boot  
threadOS ver 1.0:  
threadOS: DISK created  
Type ? for help  
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)  
-->l Test5  
l Test5  
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)  
1: format( 48 )...................freeList = 999 (wrong)  
2: fd = open( "css430", "w+" )....successfully completed  
Correct behavior of open........................2  
3: size = write( fd, buf[16] )....Pointer beyond end of filesize = -1 (wrong)  
4: close( fd )....................successfully completed  
Correct behavior of close.......................2  
5: reopen and read from "css430"..size = 0 (wrong)  
6: append buf[32] to "css430".....Pointer beyond end of filesize = 0 (wrong)  
7: seek and read from "css430"....seek(fd,10,0)=0 (wrong)  
8: open "css430" with w+..........Pointer beyond end of filetmpBuf[1]=0 (wrong)  
9: fd = open( "bothell", "w" )....successfully completed  
10: size = write( fd, buf[6656] ).Pointer beyond end of filesize = -1 (wrong)  
11: close( fd )....................successfully completed  
12: reopen and read from "bothell"size = 0 (wrong)  
13: append buf[32] to "bothell"...Pointer beyond end of filesize = 0 (wrong)  
14: seek and read from "bothell"...seek(fd,512 \* 11,0)=0 (wrong)  
15: open "bothell" with w+.........Pointer beyond end of filetmpBuf[1]=0 (wrong)  
16: delete("css430")..............Exception in thread "Thread-5" java.lang.NullPointerException  
 at FileSystem.open(FileSystem.java:96)  
 at Kernel.interrupt(Kernel.java:204)  
 at SysLib.open(SysLib.java:129)  
 at Test5.test16(Test5.java:487)  
 at Test5.run(Test5.java:51)  
 at java.lang.Thread.run(Thread.java:745)  
Exception in thread "Thread-0" java.lang.IllegalThreadStateException  
 at java.lang.Thread.start(Thread.java:705)  
 at Scheduler.run(Scheduler.java:168)  
^Clps1994@uw1-320-11:~/CSS430/FileSystemProject$ exit  
exit  
  
Script done on Wed 18 Mar 2015 07:35:02 PM PDT

## 

## Functionality Discussion

Our implementation passed two tests: open and close. Otherwise, the rest of the functionality didn’t work. This means that there is something wrong with the file table and the file table entry. We’re sure a lot of it depends on format so if formatting doesn’t work some other functionality that is dependent on it won’t. Additionally, the tests crash on the delete system call so we don’t have feedback behind that. This was due to a null access to an iNode so something must be off wherever we’re touching the total inodes.