**CSS 430: Final Project**

**Design**

*Inode.java*

Purpose: This class represents either a file or a directory. It acts as a file control block where it contains all of the information about the file. In this context, it holds 1 indirect pointer and 11 direct pointers to the index block, length of the file, flag indicating usage, and the number of file entries pointing to it.

Inode ( short iNumber ) – This is the constructor that is given the iNumber as a parameter, uses it to calculate the block number to read the corresponding block into memory, and then initializes the Inode’s data members accordingly, saving said Inode’s data with the given iNumber.

int toDisk ( short iNumber ) – This method ensures that the corresponding Inode’s information with the corresponding iNumber is written back onto the disk.

public void allocIndirectBlock ( short blockNum) – Given the block number to allocate the indirect pointer to the given block number.

public boolean deallocAllBlocks ( short iNum, Superblock superblock ) – This method deallocates all the indirect and direct blocks of the Inode with the corresponding iNumber and then writes a new empty Inode at that iNumber.

*Directory.java*

Purpose: This directory class stores and maintains the files with two arrays wherein the elements contain the size of the file and different file names.

public int bytes2directory ( byte data[] ) – We calculate the offset (incrementing by 4) which allows us to set where each block will be taking from the byte array, where each block will be set inside fsizes at the current index in the loop. We will then iterate through the fnames array to save the file name data.

public byte[] directory2bytes ( ) – We initialize a byte array that we will return to get the bytes that were converted from the Directory. By taking the fsize array and changing each corresponding block into integers, we can take the names from fnames and the file size to convert them into a temporary byte array which we will use to write its contents into our return byte array.

public short ialloc ( String filename ) – Given the filename, this method allocates a new iNumber by iterating through fsize and finding the first slot with no files in it. We then save the filename length (or set it to maxchars if it’s too big), save the filename in our fnames array, and then return its iNumber.

public boolean ifree ( short iNumber ) – This method deallocates the corresponding filename with the iNumber from the fsize array by setting it to 0.

public short namei ( String filename ) – This returns the iNumber that contains the given filename. We do so by looping through the fsize array and comparing the parameter’s length with the current element from the array as well as comparing the name itself with the current element from the fnames array. On success, it will return the iNumber, or a -1 indicating failure.

*FileTable.java*

Purpose: This class represents the file table entries using a vector. It creates a new entry which keeps track of the seek pointer and Inode number of a file.

public FileTable ( Directory directory ) – This constructor instantiates a new vector of file table entries and receives the reference of the directory.

public synchronized FileTableEntry falloc ( String filename, String mode ) – This method first looks up the filename in the directory to get the iNumber. Then it will go through a couple of test cases. If it wasn’t found but the mode was set to read, we returned null. If it wasn’t found but mode was set to write, we allocate a new file in the directory and created a new Inode. If it was found, we grab the Inode from the Disk and set the seek pointer to the end if its mode was set to append. Then, we increment the Inode’s count, and write this back onto the disk. Finally, we allocate a new file table entry in our file table and return the reference to it.

public synchronized boolean ffree ( FileTableEntry e ) – If we could successfully remove the given file table entry from our file table, we decrement its Inode count and set its flag to unused if no one else is using it, and then update this information by writing to the disk. Return true on success, and return false otherwise.

public synchronized boolean fempty ( ) – Returns a boolean indicating whether or not there are any file table entries. It will return true if the table is empty, and false otherwise.

*Superblock.java*

Purpose: This class holds data about the File System and describes the number of disk blocks, Inodes, and stores the block number of the head block of the free list.

public Superblock( int diskSize ) – Given the diskSize which represents the total number of blocks on the disk, this constructor initializes the data members with the number of blocks, Inodes, and the number of the block of the free list’s head.

public void format( int inodes ) – This method will wipe the disk of any data, resets values to their defaults, and then reformats the structure of the superblock.

public void synch ( ) – This method writes the superblock’s contents into a buffer, and then writes this information onto the disk. It is also used to ensure that the contents of the superblock are working correctly in case of any updates.

public void addToFreeList( int block ) – This method adds a freed block at the end of the freelist.

public int allocFromFreeList( ) – This method returns the next free block from freelist.

*FileSystem.java*

Purpose: The purpose of this class is to act as a layer that performs all tasks on the disk. It encapsulates all the previous classes mentioned, and instantiates them. It gives users certain operations that they can directly use such as format, read, open, close, delete, and write to a file.

public FileSystem( int diskBlocks ) – Given the disk block, it instantiates a new Super Block with that size, as well as creates a new directory and file table.

void sync( ) – This ensures that the file system is synchronized with the disk, and writes data from root to the disk, writing the directory to the disk, and then calls synch() to ensure that the super block is synchronized.

boolean format( int files ) – This deallocates all the contents from the disk and reformats it by recreating a new superblock, directory, and file table.

FileTableEntry open( String filename, String mode ) – This method opens a file corresponding to the filename that was given by calling falloc(), which retrieves the corresponding Inode from the directory.

synchronized boolean close( FileTableEntry ftEnt ) – Closes the given file table entry’s file by calling the ffree method from the FileTable class.

synchronized int fsize( FileTableEntry ftEnt ) – Returns the file entry’s corresponding Inode’s length.

synchronized int read( FileTableEntry ftEnt, byte[] buffer ) – Set a Inode to reference the FileTableEntry’s Inode, and if its flag is unused, we wait for it to be complete. Once we have acquired it, we will set it to read to prevent others from using it. We then iterate wherein we read a block at a time from the disk, and then onto the buffer. With every iteration, we determine the current block using the seek pointer, and read this into memory. Then, we will copy each byte that has been read, and increment to the next block. Once it’s complete, we notify other users that it is available, and return the number of bytes read.

synchronized int write( FileTableEntry ftEnt, byte[] buffer ) – Here, we do the same check as we do with read, but once we acquire it, we set the flag to write. We iterate through up to the size of the buffer. We determine the current block using the seek pointer and read it into memory. If the seek pointer reaches the end of the file, we allocate another block. If the block offset is greater than the length of the corresponding Inode, we make sure that there is an indirect block to read from and read the indirect block in from the disk. If the current block is set to -1, we allocate another block for the indirect to point to. We then copy each byte, write the block back to the disk when complete, and increment to the next block. Finally, we increase the file size if we wrote past the end of the file, and write this to the disk. We notify others that the file is available, and return the number of bytes written.

boolean delete( String filename ) – We grab the file we want to delete given the filename and try to open it. If we were able to close the file table entry and was able to deallocate the iNumber and the corresponding file, the method returns true.

synchronized int seek( FileTableEntry ftEnt, int offset, int whence ) – This method keeps track of the seek pointer that corresponds to the FileTableEntry that was given. It will set the seek pointer to the beginning of the file or end of the file, and will return the seek pointer upon completion.

**Output**

Connies-MBP:ThreadOS conniekim$ java Boot

threadOS ver 1.0:

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 )...................successfully completed

Correct behavior of format......................2

2: fd = open( "css430", "w+" )....successfully completed

Correct behavior of open........................2

3: size = write( fd, buf[16] )....successfully completed

Correct behavior of writing a few bytes.........2

4: close( fd )....................successfully completed

Correct behavior of close.......................2

5: reopen and read from "css430"..successfully completed

Correct behavior of reading a few bytes.........2

6: append buf[32] to "css430".....successfully completed

Correct behavior of appending a few bytes.......1

7: seek and read from "css430"....successfully completed

Correct behavior of seeking in a small file.....1

8: open "css430" with w+..........successfully completed

Correct behavior of read/writing a small file.0.5

9: fd = open( "bothell", "w" )....successfully completed

10: size = write( fd, buf[6656] ).successfully completed

Correct behavior of writing a lot of bytes....0.5

11: close( fd )....................successfully completed

12: reopen and read from "bothell"successfully completed

Correct behavior of reading a lot of bytes....0.5

13: append buf[32] to "bothell"...successfully completed

Correct behavior of appending to a large file.0.5

14: seek and read from "bothell"...successfully completed

Correct behavior of seeking in a large file...0.5

15: open "bothell" with w+.........successfully completed

Correct behavior of read/writing a large file.0.5

16: delete("css430")..............successfully completed

Correct behavior of delete....................0.5

17: create uwb0-29 of 512\*13......successfully completed

Correct behavior of creating over 40 files ...0.5

18: uwb0 read b/w Test5 & Test6...

threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)

Test6.java: fd = 3successfully completed

Correct behavior of parent/child reading the file...0.5

19: uwb1 written by Test6.java...Test6.java terminated

Correct behavior of two fds to the same file..0.5

Test completed