CSS430 - Final Project

Project Purpose

Using our new found understanding of operating systems, we were asked to implement a *file system* on the ThreadOS in java. This object oriented approach bettered our comprehension of the course material as it required for sensitive design decisions used in the classes below. The file system ultimately was comprised of adjustments and implementation within 8 core classes including the Kernel, FileSystem, Inode, Directory, TCB, FileTable, FileTableEntry, and the Superblock. Although other classes were instantiated within the ThreadOS, such as the SysLib and the Scheduler, these fundamental 8 classes were ultimately the scope of our design implementation, culminating in a File System.

Class Description, Design & Specifications

Kernel
ThreadOS boot. The kernel houses all of the thread operations necessary for the OS to function properly, as well as catching interrupts in which thread execution is halted in order to perform various functions such as reading and writing to disk, along with instantiating new threads. The kernel is essentially the master class that designates the particular order of operations for the system.

The kernel on boot instantiates our file system. System calls within the kernel pertaining to the file system include open, format, read, write, seek, close, delete, and size.

Open, read, write, seek, close, and size operations create a file table entry and manipulate the entry using the Thread Control Block. Such operations will look up a FileTableEntry from the Thread Control Block, and once found perform its respective operation on entries with the FileSystem methods.

File System
Entry point to the filesystem, abstracts out read/writes to the underlying operating system/Disk.java. The file system is the highest tier of class implementation other than the kernel, and will essentially use all of the other classes in order to perform operations on the disk and is the final step/goal in our final project implementation.

File System Life Cycle

When the OS (Thread OS) is booted, the File System will be initialized and the superblock (which describes the memory space of the File System) will be checked for validity. If the superblock has an error in its validity check, the File System will be re instantiated with a default occurrence of 64 file entries.

The root directory is given the filename “/” in order to identify it as the root. All other entries other than this root directory are considered to be contained within the root directory.

Users are able to interact with the file system with a variety of public methods.

Method	Description
public void sync()	Syncs the file system to the disk, along with the logical disk to the physical disk.
public boolean format(int files)	Creates the number of files specified by the input files value, resulting in a format of the disk.
public FileTableEntry open(String filename, String mode)	Opens a file dependent on the mode passed into the method such as “r” for read, “w” for write, “a” for append, and “w+” for a read and write. This method returns a file table entry on a successful open.
public boolean close(FileTableEntry entry)	Closes an open file.
public synchronized int fsize(FileTableEntry entry)	Returns the size of a file that is open.

public int read(FileTableEntry fte, byte[] buf)	Reads from the open file into the passed buffer from the current seek position, and the total byte amount is returned as an integer.
public int write(FileTableEntry entry, byte[] buffer)	This method writes data to the currently opened file at the seek position. The integer returned corresponds to the total number of bytes that was written.
public int seek(FileTableEntry entry, int offset, int whence)	Seeks the offset of the open file in respect to the whence value which will return either a position to the beginning of the file, end of the file, or current position within the file.
boolean delete(String filename)	Destroys the file specified by fileName. If the file is currently open, it is not destroyed until the last open on it is closed, but new attempts to open it will fail.

Inode
Inode is a pointer to a specific file or directory; primarily a data container but contains basic serialization methods to/from disk. Contains 12 pointers, of which 11 are direct and 1 is indirect.

TCB
Individually manages open file descriptors for a thread. The thread control block will be useful for retrieving and returning thread information such as the thread id and the thread itself.

FileTable & FileTableEntry
The FileTable manages file table entry allocation and deallocation; container for a group of FileTableEntry objects. The table will instantiate a list of file table entries, with each file table entry corresponding to a particular file descriptor. The file table entries will be stored in a vector as to save space on memory allocation by using a dynamic list as opposed to a predefined static array. The file table will create new file table entries and add and remove to the list of entries.

Superblock
The superblock is a key component to the Thread OS file system as this block describes the memory space of the file system. This class reads from the disk and provides information including the number of free blocks available within the disk. The superblock communicates and

read and write its contents back to the dick. The disk block will contain information such as the number of inodes, head block of free list.

Directory
Single root directory for the filesystem, methods allocate and deallocate additional inodes for files. The directory essentially holds, as well as manage the different files that are being manipulated within the system by creating two arrays to hold the file name size along with the file name. The directory not only allocates a new Inode number but performs functions such as searching for files, along with reading and writing between the directory and byte array.

Within the Directory constructor the number of Inodes/maximum number of concurrently openable files is taken in, storing a file entry for each each file in the filename and filesize arrays.The fname array contains the inode along with the filename two-dimensionally. The size of the file is contained in the fsize array, at the same index of the Inode and filename within the fname array.

The directory contains the following methods including

Method	Description
public void bytes2directory(byte data[])	Accepts a byte array containing Directory data stored on disk. Initialized the Directory instance with data from the input array.
public byte[] directory2bytes()	Coverts and returns the Directory data in the form of a byte array that will be written back to disk.
public short ialloc(String filename)	Accepts a filename as input and allocates an inode number for new file.
public boolean ifree(int iNumber)	Accepts an inode number and deallocates the corresponding file. The corresponding file will be deleted.

public short namei(String filename)	Accepts a filename and returns the corresponding filename. Returns an error code if the filename is not in the directory.

File System Limitations

- No file name can be longer than 30 characters
- Maximum size of a file is 136,704 bytes
- There is only one directory within the file system - the root directory "/"
- Files cannot be deleted if they are currently opened.
- Only information contained regarding a file is the size, as ownership and permission metadata is disregarded.
- Does not contain the creation and/or modification times of the files.

Analysis

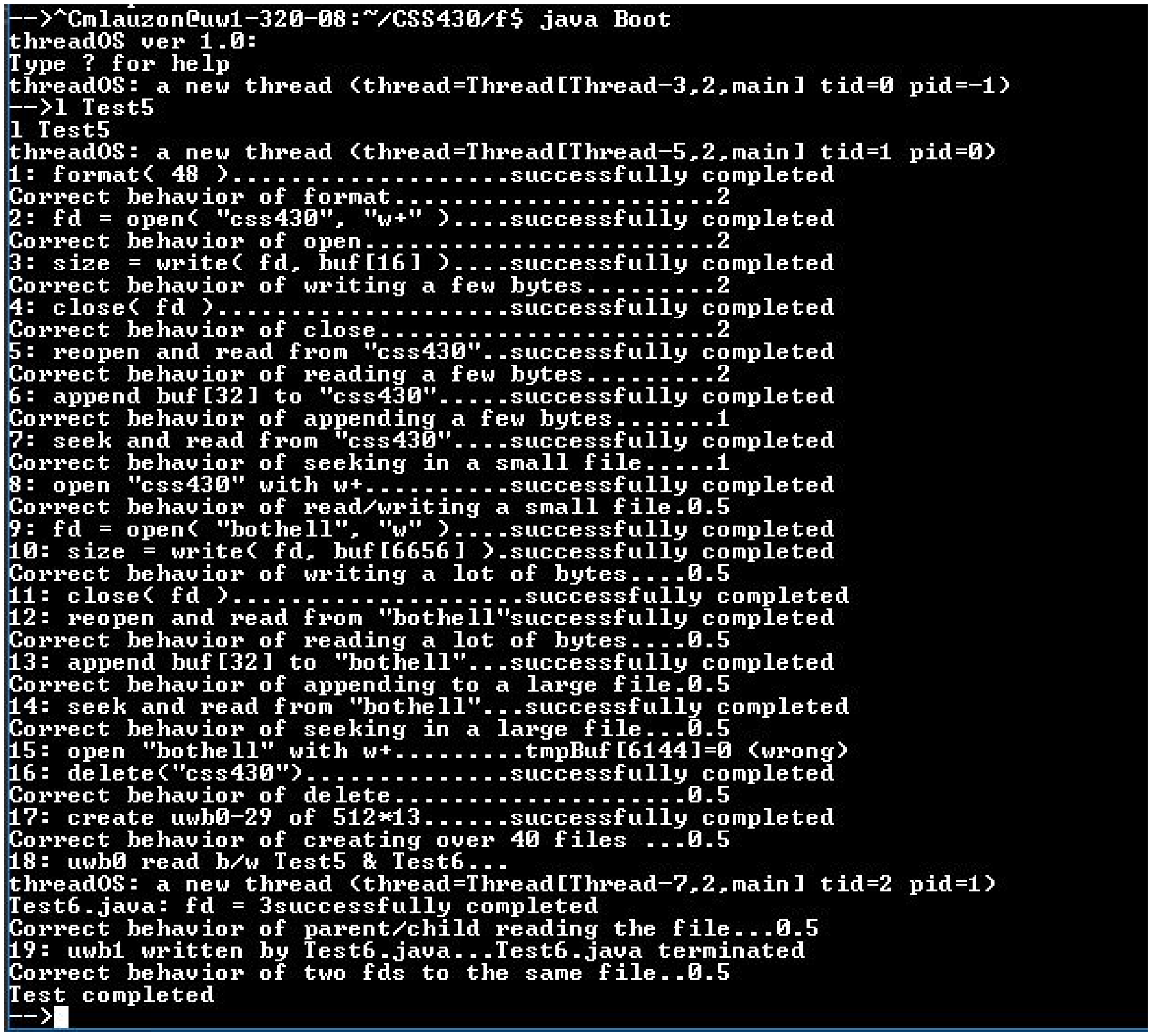
Since our group implemented an indexed allocation scheme in regard to block accessing, we were able to achieve a linear lookup time (O(1)) of any raw block identifier given its current offset value in a file.

In terms of performance, we did not measure quantifiably how ThreadOS performed. Techniques such as external timing, to gather performance information would have outside dependencies that introduce variations in results due to differing hardware. Furthermore, due to time constraints, we omitted independent testing of ThreadOS from the file. As a result, most of our analysis derives from using Test5 provided by the instructor. All but one test pass in our program.

If we were to implement the File System again, we would implement caching in order to allow a preliminary period between read and writes. Each Inode having to be written to disk regardless of the shared raw blocks existing between them, results in a decrease in performance time. More writes are required as opposed to saving an intermittent locality in which this data could be cached, resulting in less reads as each write operation necessitates a read of the faw block prior to writing in order to check the boundary and size of the current block.

Test Results

Our file system passed nearly all of the tests contained within the Test5.java file, with the exception of Test 15.



```
-->^Cmlauzon@uw1-320-08:~/CSS430/f$ 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+.........tmpBuf[6144]=0 (wrong)
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
-->
```