**CSS 430: OS Final Project File System**

**Winter 2018**

**Group members: Natnael Tereza, Tyler Do, Artiom Voronin**

**The following files were modified/added:**

* Directory.java
* FileSystem.java
* FileTable.java
* Inode.java
* Kernel.java
* Scheduler.java
* SuperBlock.java
* TCB.java

**Description of Internal Design**

***Kernel.java***

Kernel fulfills calls made by SysLib. It also initializes a File System object upon BOOT system call and uses it since a lot of calls require to interact with FileTableEntry that represents a file in a memory.

**Added data Members:**

private static FileSystem fs

private final static int COND\_DISK\_REQ = 1;

private final static int COND\_DISK\_FIN = 2;

**System calls added/modified:**

* OPEN: opens the file specified by the fileName string
* CLOSE: closes the file corresponding to fd
* SIZE: returns the size in bytes of the file indicated by fd
* SEEK: updates the seek pointer corresponding to fd
* FORMAT: formats the disk
* DELETE: deletes the file specified by fileName
* SYNC: synchronized memory data to disk data
* READ: reads up to buffer.length bytes from the file indicated by fd
* WRITE: writes the contents of buffer to the file indicated by fd

***Scheduler.java***

**Methods added/modified:**

* addThread
* deleteThread

***TCB.java***

Each user thread maintains a user File Descriptor Table in its own TCB. The class basically maps a connection between file descriptor number and file table entry.

**Added data Members:**

public FileTableEntry[] ftEnt

**Methods added/modified:**

* public synchronized int getFd: add FileTableEntry object and return its index
* public synchronized FileTableEntry returnFd: return FileTableEntry and reset it
* public synchronized FileTableEntry getFtEnt: return FileTableEntry

***FileTableEntry.java***

This class bundles up the seek pointer, iNode reference, mode, count, iNumber.

**Added data Members:**

public int seekPtr

public final Inode inode

public final short iNumber

public int count

public final String mode

**Methods added/modified:**

FileTableEntry ( Inode i, short inumber, String m ) : constructor

***FileSystem.java***

This class ties everything together between all the other classes (Kernel, TCB, FileTableEntry, FileTable, Inode, SuperBlock, Disk, Directory).

It is responsible for manipulating disk. It provides user threads with system calls that will allow them to format, to open, to read from, to write to, to update the seek pointer of , to close, to delete, and to get the size of their files. It hides all the implementations from user threads.

**Added data Members:**

private SuperBlock superblock

private Directory directory

private FileTable filetable

private final int SEEK\_SET = 0;

private final int SEEK\_CUR = 1;

private final int SEEK\_END = 2;

**Methods added/modified:**

public FileSystem(int blocks): Constructor, initializes superblock, directory and filetable

public void sync(): Syncs the file system back to the physical disk. Makes sure the superblock is synchronized and writes the directory information onto the disk in byte form.

public boolean format(int files): Formats the disk by creating a new superblock, directory and file table onto the disk.

public FileTableEntry open(String fileName, String mode): Responsible for opening a file specified by fileName string in the also specified mode string. If the mode is in “w”, then it will just return null.

public boolean close(FileTableEntry entry): Responsible for closing the file corresponding to the given file table entry. Returns a boolean value upon either a successful close or a failed one.

public synchronized int fsize(FileTableEntry fd): Returns the file size in bytes indicated by the fd

public int read (FileTableEntry fd, byte[] buffer): Reads up to buffer.length bytes from the file indicated by fd starting at the position currently pointed to by the seek pointer. If bytes remaining between the current seek pointer and the end of file are less than buffer.length, SysLib.read reads as many bytes as possible, putting them into the beginning of buffer. It increments the seek pointer by the number of bytes to have been read. The return value is the number of bytes that have been read, or a negative value upon an error.

public int write(FileTableEntry fd, byte [] buffer): Writes the contents of buffer to the file indicated by fd. The operation may overwrite existing data in the file and/or append to the end of the file. SysLib.write increments the seek pointer by the number of bytes to have been written. The return value is the number of bytes that have been written, or a negative value upon an error.

public int seek(FileTableEntry fd, int offset, int location): Updates the seek pointer corresponding to fd as follows. If whence is SEEK\_SET (= 0), the file's seek pointer is set to offset bytes from the beginning of the fileIf whence is SEEK\_CUR (= 1), the file’s seek pointer is set to its current value plus the offset. The offset can be positive or negative. If whence is SEEK\_END (= 2), the file's seek pointer is set to the size of the

file plus the offset. The offset can be positive or negative.

public boolean delete(String fileName): Deletes the file specified by fileName. All blocks used by the file are freed if the file is not being currently used. If the file is deleted, then returns a 0, otherwise returns a -1.

private boolean deallocAllBlocks(FileTableEntry fileTableEntry): this is a helper method helping to deallocate all blocks

***FileTable.java***

This class is shared among all user threads that keep track of all the files in the system. Its basic function is to move iNode on/off the disk and wrap it into FileTableEntry.

**Added data Members:**

private Vector table

private Directory dir

short USED = 0

short UNUSED = 0

**Methods added/modified:**

public FileTable( Directory directory ): Constructor, initializes table and dir

public synchronized FileTableEntry falloc( String filename, String mode ): Allocates a new file (structure) table entry for this file name. This method will allocate/retrieve and register the corresponding inode using dir. Depending on what the mode is, it will increment the inode count and then will return the file table entry.

public synchronized boolean ffree( FileTableEntry e ): Frees the file table by receiving a file table entry reference and going through the cache. If the fte is found in the table, the method returns true.

public synchronized boolean fempty( ): Returns a boolean value if there is an entry in the File TABLE.

***Inode.java***

This class helps to keep track of which blocks are in the file. Each iNode tracks one file in the system. iNode has 12 pointers, 11 of them are direct and 1 is indirect.

**Added data Members:**

private final static int iNodeSize

public final static int directSize

private final static int maxBytes

public int length

public short count

public short flag

public short direct[]

public short indirect

**Methods added/modified:**

public Inode( ): Default constructor, initialize all members to empty values

public Inode( short iNumber ): Constructor, initialize using iNumber

public void toDisk( short iNumber ): save to disk as the i-th node

public int findIndexBlock(): Returns indirect pointer.

public boolean registerIndexBlock(short iNumber): Set indirect pointer. If the indirect is not -1 or if the direct pointers are -1 then it will return false. Otherwise it will set the indirect to the iNumber and then return true after finishing writing its data.

public int findTargetBlock(int offset): Find a block based on the offset depending on the following conditions. If the target is less than 0 return a -1, if the target is less than 11 than return the direct of the target. Otherwise use the bytes2short to calculate block size and then return it.

public int registerTargetBlock (int entry, short offset): Get index of a block

public byte[] unregisterIndexBlock(): Free indirect block

public boolean setIndexBlock(short blockNumber): Set indirect block to block number.

***SuperBlock.java***

This class manages the list of free blocks, number of disk blocks, and number of inodes. It also represents the first block of the disk.

**Added data Members:**

private final int defaultInodeBlocks

public int totalBlocks

public int inodeBlocks

public int freeList

**Methods added/modified:**

public SuperBlock(): Default constructor, intializes data members to empty values

public SuperBlock( int diskSize ): Reads the SuperBlock from disk and initializes the member variable for the diskSize.

public void format(): Calls the format(int) method to clean the disk.

public void format(int numFiles): Cleans the disk and resets the correct structure if there

is illegal state during initialization of an instance.SuperBlock instance variables are cleared to default values and written to the recently cleared disk.

public void sync(): brings the physical SuperBlock contents inline with any updates made to the Superblock class instance.

public int getFreeBlock(): returns next free block location

public boolean returnBlock(int paramInt): add block to freeList

***Directory.java***

This class represents a single level file system. It provides the directory entries for the file to be named and obtain size.

Added data Members:

private static int maxChars

private int fsizes[]

private char fnames[][]

Methods added/modified:

public Directory( int maxInumber ): Constructor, intializes fsizes and fnames.

public void bytes2directory( byte data[] ): Initializes the Directory instance with data. For each file size entry, it will convert the byte data into integers by an offset of a block size. The converted data is stored in the fsizes array index. The converted data is then given a name that holds the data, offset and number of chars. The method ends with us getting the chars at the specified location.

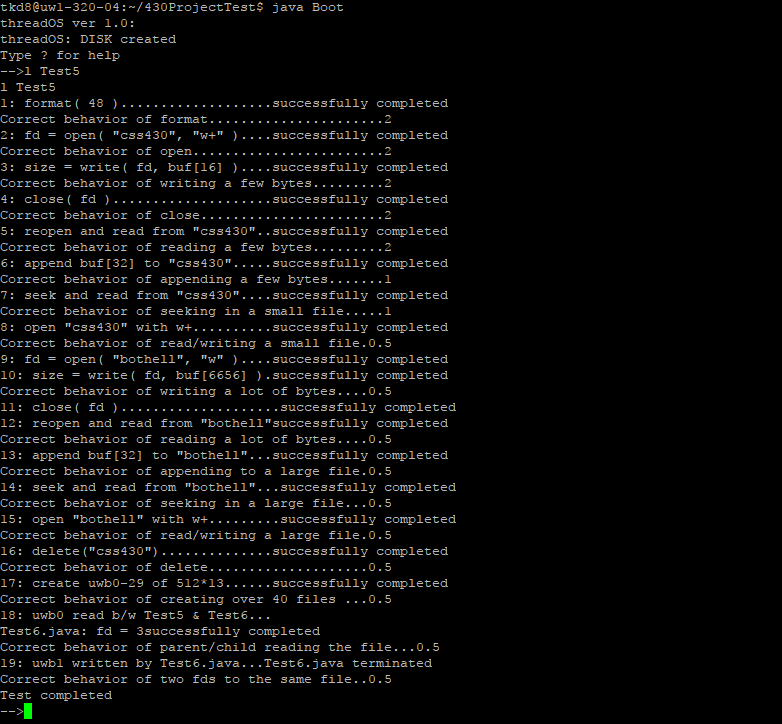
public byte[] directory2bytes( ): Converts and return Directory information into a plain byte array. After it is converted, the byte array is written back onto the disk. Similar to the above method with converting data and then storing it into a string. The method ends with us copying the bytes array into the byte array “dir”, which is then returned.

public short ialloc( String filename ): allocates a new inode number for this filename

public boolean ifree( short iNumber ): deallocates inode number

public short namei( String filename ): returns the number representing the filename

**Result (Test 5)**



**Performance, Assumptions and Limitations of the FileSystem**

The end result of the project is unix-like file system on ThreadOS. Through the use of the file system, user programs are able to access persistent data on disk by way of stream-oriented files rather than the more painful direct access to disk blocks with rawread() and rawrite().

The file system contains a limited number of the system calls implemented to work with.

The maximum number of block in the disk is 1000 blocks.

Assumes that the given test file is a valid file to run with our system

Each block is of size 512 bytes.

The maximum number of iNodes is 64.

Each iNode is 32 bytes and has 11 direct and 1 indirect pointers.

All files are created and supported under the single-level, root directory.

**Possible Extended Functionality**

* One of the ways to extend functionality of our File System is to add support for multi-level directory, since currently all our files reside and are supported in the root directory.
* We can increase the number of files the FileSystem can process at a time by increasing the FileTable Size.
* Our current ThreadOS is not capable of creating new file. We could make it create new files.
* We can also make the FileSystem capable of erasing the file content while keeping the file attributes.