Final Report

Kuo Ting Shih, John Walter

File System Specification, Assumption, and Limitations

FileSystem.java

FileSystem class implements basic functions to manage the the filesystem related calls coming down to it from Kernel interrupts. There are eight basic functions in this class that correspond with system calls originating in SysLib and forwareded from Kernel. FileSystem is limited in its functionality in that not all file related functions are supported, such as permissions.

Directory.java

Upon booting ThreadOS, the file system instantiates the Directory class as the root directory through its constructor, reads the file from the disk that can be found through the inode 0 at 32 bytes of the disk block 1, and initializes the Directory instnace with the file contents. The file system will write back the information on Directory to the disk when shutting down ThreadOS. Directory is limited (and frankly unrealistic) in that there is only one flat directory with all files in it. Virtually no modern filesystem follows this pattern.

FileTable.java

FileTable is shared among all user threads in the file system. The FileTable class manages FileEntry objects. When a user thread allocates a new entry of the user file descriptor table, the entry number itself becomes a file descriptor number. The entry maintains a reference to a file table entry. The user thread then requests the file system to allocate a new entry of the filetable maintained by the system. The file system finds the corresponding inode and records it in this file table entry. The user thread then registers a reference to this file table entry in its own descriptor table entry.

Inode.java

There are 12 pointers in Inode class. 11 pointers are direct pointer to a block and 1 pointer is indirect pointer to a block. An inode represents the metadata that describes a file in the system. The Inode class members consist of filelength, number of table entries, and a status flag.

SuperBlock.java

SuperBlock class represents the first block of the disk, which is managed by ThreadOS. It has the information of number of free disk blocks, number of files, freelist head pointer, and number of total disk blocks.

Internal Design Description

FileSystem.java

Constructor(s)

- FileSystem(int diskBlocks): initializes SuperBlock, Directory, and FileTable members

Members

- SuperBlock superblock: the SuperBlock object in the FileSystem.
- Directory directory: the Directory object in FileSystem.
- FileTable filetable: the FileTable object in FileSystem.

Methods

- boolean format(int files): formats the filesystem based on the number of files indicated by the files parameter.
- FileTableEntry open(String filename, String mode): opens a file matching the filename in the first parameter with the access mode in the second parameter.
- boolean close(FileTableEntry ftEnt): decrements the number of user-occurances of the file open in the system-wide filetable. If the count is zero, removes the filetable entry from the filetable.
- boolean delete(String filename): attempts to close a file based on its inode number and remove it from the directory.
- int seek(FileTableEntry ftEnt, int offset, int whence): positions the seek pointer within a filetable entry based on the parameterized offset and seek mode.
- int read(FileTableEntry ftEnt, byte[] dirData): reads a file that has either the “r” or “w+” access mode by looking up the disk data specified by the inode number in the parameterized filetable entry and copies the read data into a buffer. Maintains the seek pointer and returns the number of bytes read.
- int write(FileTableEntry ftEnt, byte[] dirData): writes to a file that has either the “w”, “w+” or “a” access mode as appropriate by the mode by first finding the block to write to and then copying the parameterized data into the block on disk and returns the number of bytes written.
- int fsize(FileTableEntry ftEnt): simply returns the size of the filetable entry.
- boolean deallocAllBlocks(FileTableEntry ftEnt): attempts to deallocate all blocks associated with the parameterized filetable entry.
- void sync(): commits the entire contents of the directory to disk and syncs the superblock.

Directory.java

Constructor

- Directory(int maxInumber): initializes the fsizes and fnames members.

Members

- fsizes: an array of integer which stores file sizes.
- fnames: a 2D array of char which stores the file names of each element.

Methods

- void bytes2directory(byte[] data): Assume receiving directory information from disk and initialize the Directory instances based on data[].
- byte[] directory2bytes(): converts and returns Directory information into a plain byte array. This byte array will be written back to disk.
- short ialloc(String filename): allocates a new inode number for the file and create the file based on the parameterized filename.
- boolean ifree(short iNumber): deallocates and deletes the file based on iNumber.
- short namei(String filename): returns the inumber corresponding to the parameterized filename.

FileTable.java

Constructor

- FileTable(Directory directory): initializes the table and dir members.

Members

- Vector table: a table of filetable entries.
- Directory dir: the filesystem directory

Methods

- synchronized FileTableEntry falloc(String filename, String mode): allocates a new filetable entry to the filetable based on the parameterized filename and mode and returns the filetable entry.
- synchronized boolean ffree(FileTableEntry ftEnt): removes a filetable entry from the file table and decrements the number of inode references to it.
- synchronized boolean fempty(): simply returns whether or not the filetable is empty.

Inode.java

Constructors

- Inode(short iNumber): initializes all members according to the parameterized iNumber
- Inode(): initializes all members to empty values.

Members

- int length: file size in bytes
- short count: number of filetable entries pointing to this inode
- short[] direct: direct pointers
- short indirect: indirect pointer
- short flag: 0 = unused, 1 = used ...

Methods

- int findTargetBlock(int offset): finds a target block based on the parameterized offset.
- int setTargetBlock(int offset, short targetBlock): adds a block to an inode.
- byte[] freeIndirectBlock(): reads the data pointed to by the indirect pointer and resets the pointer.
- boolean setIndexBlock(short iNumber): sets the indirect pointer if all direct pointers are taken and the indirect pointer is free.

- void toDisk(short iNumber): commits the block associated with the parameterized iNumber to disk.

SuperBlock.java

Constructor
- SuperBlock(int diskSize): initializes the superblock

Members
- int totalBlocks: total number of blocks
- int totalInodes: total number of inodes
- int freeList: pointer to the first free block

Methods
- void format(int inodes): formats the disk with the number of inodes indicated by the parameter.
- getFreeBlock(): removes a block from the free list and returns it.
- boolean returnBlock(int blockNumber): adds a block back to the free list.
- void sync(): commits all of the superblock information to disk.

Consideration on performance estimation, current functionality, and possible extended functionality

The first and the last tests were noticibly slow. All other tests were reasonably fast, although still not as instantaneous as what one would normally expect from a filesystem.

ThreadOS is not an ideal operating system for the simple fact that it is a simulated OS being run on a Virtual Machine being run on an actual OS. The fact that there are being made system calls into a simulator that makes system calls that make system calls is doomed to return poor performance.

There are many features that one could think of that are not being implemented in this system. Permissions were mentioned above. A flat directory structure is probably the most glaring of absent features, and with that would come the ability to move or copy files. Other metadata about files in the filesystem could include the time and date a file was created or last edited, and enforcement of file extensions (on operating systems that support or enforce this). And of course, there is no GUI associated with this filesystem or with ThreadOS itself.

Test5 results