Uzair Ahmad & Mankirat Dhaliwal

Prof. Munehiro Fukuda

CSS 430

Winter 2023

Final Project Report

Specification

The overarching theme and highest level class of this project was the File System class, as we were to build a single-level Linux-like file system based on ThreadOS. This class serves as an interface through which users can interact with the file system directly. The goal was to create a filesystem that would allow user programs to access persistent data on disk via stream-oriented files rather than direct access to disk blocks. The system is created in this program was a single-level system, with the file system defining the root directory. It was always open for user threads to save their files.

The class was built using a skeleton file provided in the assignment description. Because a file can do the following actions: format, open, close, fsize, read, write, delete, and seek, these were the methods implemented in the file system class. In addition, because a file system included a superblock, a directory, and a filetable, the file system class was dependent on the completion of other classes (which contain entries). As a result, we decided to implement this class last because we needed the other classes and some of their functions. The file system would be instantiated in Kernel.java as well because when the system boots up, the file system would be booted up with 1000 disk blocks, as we learned in class.

As previously stated, the content and structure of this class were based on the skeleton file provided. As a result, the approaches we developed were not entirely based on our own ideas, but rather on what was provided. This made sense to us as the only method we would require because a filesystem is only allowed to do certain things, while other activities must be performed in the dependent classes, which is why the filesystem instantiates these dependent classes.

Data Members:

dir (private Directory): a reference to the root directory of the file system.

superblock (private SuperBlock): an instance of the SuperBlock class that represents the file system's superblock.

filetable (private FileTable): an instance of the FileTable class that represents the file system's file table.

Constructor:

FileSystem(int diskBlocks): initializes the file system by creating a new SuperBlock with diskBlocks blocks, creating a new Directory with the total number of inodes from the superblock, and creating a new FileTable with the root directory.

Methods:

read(FileTableEntry ftEnt, byte[] buffer): reads data from the file represented by ftEnt and stores it in the buffer. The ftEnt object contains the seek pointer, mode, and inode number for the file being read. If the mode is "w" or "a", the method returns -1 to indicate an error. The method reads data in blocks from the disk and copies it into the buffer. It updates the seek pointer and returns the number of bytes read.

format(final int n): formats the file system by deallocating all files in the file table and re-initializing the superblock, directory, and file table. The method returns true if the operation was successful.

write(FileTableEntry ftEnt, byte[] buffer): writes data from the buffer to the file represented by ftEnt. The ftEnt object contains the seek pointer, mode, and inode number for the file being written. If the mode is "r", the method returns -1 to indicate an error. The method writes data in blocks to the disk and updates the seek pointer and inode length as needed. It returns the number of bytes written.

fsize(FileTableEntry ftEnt): returns the size of the file represented by ftEnt in bytes.

deallocAllBlocks(FileTableEntry ftEnt): deallocates all the blocks used by the file represented by ftEnt and updates the inode. The method returns true if the operation was successful.

open(String filename, String mode): opens the file with the given filename in the specified mode (read, write, or append) and returns a new FileTableEntry object that represents the file. If the file does not exist and the mode is "r", the method returns null. If the mode is "w", the method deallocates all blocks used by the file and creates a new empty file. If the mode is "a", the method opens the file for writing at the end of the file.

close(FileTableEntry ftEnt): closes the file represented by ftEnt and updates the file table. If the file was open for writing, the method updates the inode's last modified time. The method returns true if the operation was successful.

Descriptions on internal design

SuperBlock: The SuperBlock class represents the superblock of a disk. It has three instance variables: totalBlocks, totalInodes, and freeList. The constructor of the class takes a diskSize parameter and initializes the instance variables by reading the first block of the disk using the rawread() method of the SysLib class. If the totalBlocks is equal to the diskSize, totalInodes is greater than 0, and freeList is greater than or equal to 2, then the constructor returns. Otherwise, it sets the totalBlocks to diskSize and calls the format() method to format the disk.

The sync() method of the class writes the updated superblock data back to the disk using the rawwrite() method of the SysLib class.

The format() method of the class initializes the disk with a specified number of inodes. It takes the files parameter as the number of inodes to be created. If the files parameter is less than 0, then the files parameter is set to 64 (default number of inodes). The method initializes each inode to default values and fills the blocks with empty data. It also updates the free list and writes the updated data to the disk using the rawwrite() method.

The getFreeBlock() method of the class returns a free block from the disk. It reads the block data from the disk using the rawread() method and updates the free list by setting it to the next free block. It then writes the updated block data back to the disk using the rawwrite() method and returns the location of the free block.

The returnBlock() method of the class returns a block to the free list. It writes the block data to the end of the free list and updates the free list by setting it to the block number.

Directory: The "Directory" class in Java is used to create and manipulate a data structure that represents a file directory. It contains several instance variables, including "maxChars", "fsize", "fnames", and "size". "maxChars" is an integer that specifies the maximum number of characters allowed in a file name, while "fsize" is an integer array that stores different file sizes. "fnames" is a two-dimensional character array that stores different file names, and "size" is an integer value that indicates the maximum number of files.

When the Directory class is initialized, the constructor sets all file sizes in the "fsize" array to 0, and initializes the "fnames" array with the root directory "/".

The Directory class includes two methods, "bytes2directory" and "directory2bytes", for converting the file directory between byte array and directory formats. The "bytes2directory" method takes a byte array as input and parses through it to extract file sizes and names, while the "directory2bytes" method converts the directory into a byte array format that can be stored on a disk.

The "ialloc" method is used to allocate an inode to a new file and returns the inode number. On the other hand, the "ifree" method frees an inode and returns "true" if it is successful. Finally, the "namei" method is used to return the inode number for a given file name.

Overall, the Directory class provides a basic implementation of a file directory data structure that can be used in a file system. It allows for the allocation and freeing of inodes, and the conversion of the file directory between different formats.

FileTable: This Java class is called "FileTable" and it represents a table of files in a file system. When the class is constructed, it creates a Vector to store the file entries and receives a reference to the root directory of the file system. One of the important methods of this class is called "falloc". It is a synchronized method that takes a filename and mode as inputs and returns a new FileTableEntry object for that file. It performs several checks and conditions to allocate or retrieve the corresponding inode for the file and then adds a new FileTableEntry object to the table. The falloc method is synchronized to prevent multiple threads from accessing or modifying the file table at the same time.

Another important method provided in this class is called "ffree". It is also a synchronized method and is used to remove a specified FileTableEntry from the table and release its associated inode. The ffree method checks the flag of the inode to see if it is being used, and updates the flag and count values accordingly. It also updates the inode on disk and notifies waiting threads that the inode is available. The method returns a boolean indicating whether the specified entry was found in the table or not.

Lastly, the class also includes a synchronized method called "fempty". This method returns a boolean indicating whether the table is empty or not. This method can be useful for checking if the table is ready for formatting.

TCB: The TCB class is a thread control block that provides a way to manage and track threads in an operating system. It has private instance variables to hold the thread, its unique identifier (tid), the identifier of its parent thread or process (pid), a boolean flag indicating whether the thread has terminated, an integer representing the number of milliseconds that the thread should sleep, and an array of FileTableEntry objects for file system operations. The class constructor initializes the instance variables using the parameters provided. Getter and setter methods for the private variables ensure thread safety. The TCB class also has methods for file system operations, including adding and removing FileTableEntry objects from the ftEnt array and returning the corresponding FileTableEntry object from the ftEnt array. Overall, the TCB class provides an essential component for managing threads and performing file system operations in a thread-safe manner.

Consideration of execution performance and functionality

We needed so many methods in the superblock class since it cannot be accessed by any user thread and so can only be controlled by the operating system. As a result, having its own ways to function was required, as was having its own information. Basically, it was a reasonably simple class because it represented the system's 0 block. It is analogous to the Unix volume control block.

The Directory class is configured using the skeleton file provided in the assignment description. The skeleton file contained a few methods that needed to be implemented. Because the directory is designed to contain and manage the files in the system, having these ways makes sense because some of the methods were simply converting the directory to bytes or bytes to directory, making it easier to store and handle things. As previously stated, there are numerous characteristics in this class to keep track of all the different items the directory maintains because it is responsible for holding the files and their names. Therefore, they need to be stored in the directory. As we can see in the description above of the directory class, it is a pretty straightforward class and is like a normal directory we would think of in an OS.