Brody Schulke && Christopher DuBois

CSS 430 Parsons

File System Project

March 12th, 2016

Testing Results

```
threadOS ver 1.0:
threadOS: DISK created
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Test5
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
l Test5
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
-->quit
quit

Process finished with exit code 1
```

Specification of Assumptions and Limitations

We are limited by the fact that we are following ThreadOS's static file sizes, number of inodes, the specified options of file reading and writing. We are stuck with the max block size of 512 bytes. Another limiting factor includes the restriction of ThreadOS's methods and implementations. As a specification, the project is implented so the following communication channel occurs: SysLib → Kernel → FileSystem → SysLib's rawread and rawwrite. Within FileSystem, there are many calsses that interact with one another to perform the desired functionality of the File System. The FileSystem itself is limited to a single root directory, and does not allow for a recursive file structure. The assumptions we made for the project are simply that the restrictions on implementation are the same as those specified throughout the project description, and that tests are also following that same assumption. It also assumed that Test5 is the main test case that tests all of the proper functionality of the File System as a whole.

Descriptions on Internal Design

Inode

The Inode is a simplified version of the Unix Inode but retains the functionailty of describing files. The Inode contains 12 pointers, 11 of which are direct blocks, while the 12^{th}, is an indirect pointer to another set of blocks. To ensre that there are no inconsistencies, before inodes are updated they are checked on disk to see if they have been updated, and if so, the contents are written back to the disk immediately. The iNode specified in this project is limted by 32 bytes to a file, and has each block is 512 bytes. The iNode stores a length, count, flag, in addition to the direct and indirect pointers, the length is meant to describe the file size in bytes, the count is for the number of file table entries in the iNode, and the flag is to indicate whether or not the inode is 0=unused, 1 =used, 2=reading, 3=writing, 4=read and write and 5 is for appending. The Inode will create a new file for options 3, 4, 5 if the file does not exist. An Inode is initialized as a blank inode by grabbing the inode number and using this to extract the block number from syslib, initializing where the direct pointera dn indirect pointers, by utilizing SysLib's rawread, and bytes2short and bytes2int methods. iNodes method is defined by writing to disk, finding the target block, registering an index block, registering a target block, and unregistering an index block. Respectively, these methods essentially utilize methods such as rawread, rawwrite, int2bytes, bytes2int, and keeping track of the direct and indirect access pointers to properly describe the file and perform called file operations.

Directory
Directory is the representation of a file cabinet as it has only one folder, the root directory. It is an abstracted class that will allocate the slots for files, locate free slots in the directory for files, and finding particular files for a user. This is done through interacting with inodes inumber values acting as an index for the location of the file and to perform file operations in that location. The directory also provides an API for converting an entire directory into a stream of bytes, and vice versa, converting a full stream of bytes into a directory. The directory is a simple abstraction, maintaining an array of file names and an array of file sizes. The directory mostly utilizes SysLib's bytes2int, int2bytes, and various array methodologies to accomplish it's tasks.

FileTable
The FileTable maintains the file entries in the file cabinet. This is accomplished through maintaing a vector of file table entries, and a reference to the root directory. The FileTable maintains three core functions, to allocate a file through falloc, to free a file through ffree, and to notify a user if the table is empty or not (which is often used for formatting). Falloc is responsible for allocating a new file table entry based on the given filename and requested file mode (such as read, write, etc). It does so by initializing a new inode and setting it's file operator flags. Falloc also initializes the file table entry and the inode is written to disk, and finally the table adds the element to it's structure. Ffree is similar although it does the exact opposite, it pulls the file table entry by decrementing the inode count and setting the inode flags. Ffree is essentially responsible for freeing the file table entry from file table's memory.

SuperBlock
SuperBlock is the first disk block and used to describe various aspects of the whole file system and OS. It describes the number of disk blocks, the number of inodes, and the block number of the head block of the free list. The constructor initializees the data of the data of a SuperBlock using a given argument which represents the number of blocks on disk. Then, it assigns values to represent the three describing aspects of the file system and OS through converting the byte data and reading from the disk into integers and assigning appropriately. It also ensures that the disk is formatted correctly. SuperBlock defines the methods sync, format, get the first free block, and add a free block to the free list. Sync stores the SuperBlock's contents to disk. Format cleans the disk and completely formats it to its appropriate form if any invalid contents of the disk are detected by the SuperBlock. The first free block is returned from the list

where the free list's block is at the very top. The value of the Free List is updated by converting the byte data from the disk and assigning its value. A freed block is added to the free list by creating a byte array and initializing all values to 0 and then adds the freed block. Right after it writes the block to disk and assigns the location of the freeblock to the free list.

FileSystem

The FileSystem is the main abstraction that pieces together all of the other parts for this project. It maintains the OS managed SuperBlock, holds the directory and the file table. The FileSystem is initialized with the number of disk blocks for the system which is passed to the OS managed superblock for full object initialization. The root directory of the file system is created with the number of inode blocks and the file table is created through the blank directory. FileSystem provides the interface for file i/o operations, such as sync, format, open, close, read, write, delete, seek. Sync writes the file table entry back to disk and syncs the super block. Format actually completely reformats the file system by resetting the file table, directory and the superblock. Open is a simple operation that allocates a new entry in the file table with the specified file name and mode of access. Open handles for concurrent read/write access by not allowing readers to access while writing. Close is the exact opposite of open, it frees a file table entry from the file table while ensuring no other file threads are accessing that file table entry at the same time. Read and write are both done recursively by incrementing a seek ptr through the bytes of the file as reads/writes occur. Delete is used to delete a file given a specific file name to search for in the directory by freeing the specified inumber from the directory and closing the file. Finally, the last methodical operation available in FileSystem is seek. Seek is used for randomly accessing locations in a file as if the file was a byte channel. This allows a user to specify where in the file they want to begin their cursor to read from. It also allows a user to skip around the file by given index values as long as the index values are not less than zero or greater than the file length. Seek is an important operation in the FileSystem as it allows users to jump through the file.

Kernel

kernel was altered to accommodate for the various functionalities required by the file system itself. It can accommodate for read, write, open, close, size, seek, format, and delete. All of these methods directly translate to those named similarly in the FileSystem and are implemented by directly calling those methods within FileSystem with the various parameters that were passed through by the SysLib call.

SysLib

SysLib was altered in order to send proper Kernel interrupts with the specified program method calls required: format, open, close, read, write, seek, fsize and delete. The methods simply call Kernel interrupts at statically specified integer values, as well as passing in the parameters the SysLib methods themselves receive to the kernel, which will then appropriately delegate and dispatch filesystem methods.

Consideration on Performance and Functionality

Functionality-wise, the File System project does not provide the same extent of a typical modern file system API that an OS implements. It is limited to a smaller subset, and as such, doesn't maintain the higher level of functionality but still gets the point across. The point gets accross with mediocre efficiency and key implementation methods such as seek, read, write, append, creating a file, saving to disk etc. However, its performance its relatively good considering its situation of being a simulated OS on top of a virtual machine on top of another OS, creating a lag of communication between all of these levels. However, ThreadOS is threadded so it allows for concurrent access to the File System. For extended functionality, a user should be able to include subnested directories, have an API for compressing file systems, and a friendly user-interface. The restrictions on sizes of various files including file names should also be lifted to allow for larger sets of files to be included in the File System. All in all, the file system, as a miniature, simulated version, is a clear and functional (semi-performant) program encapsulated inside of ThreadOS.

Linux compilation and Run

```
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
-->quit
quit
```