CSS 430 Project 5 Report

Group 7:
Yeseul An
Yingjun Fan
Lei Wu
6/9/2019

Table of Contents

Purpose

The purpose of this project is to build a Unix-like file system on the ThreadOS. User programs are be able to access persistent data on disk by way of stream-oriented files rather than direct access to disk blocks with rawread() and rawwrite().

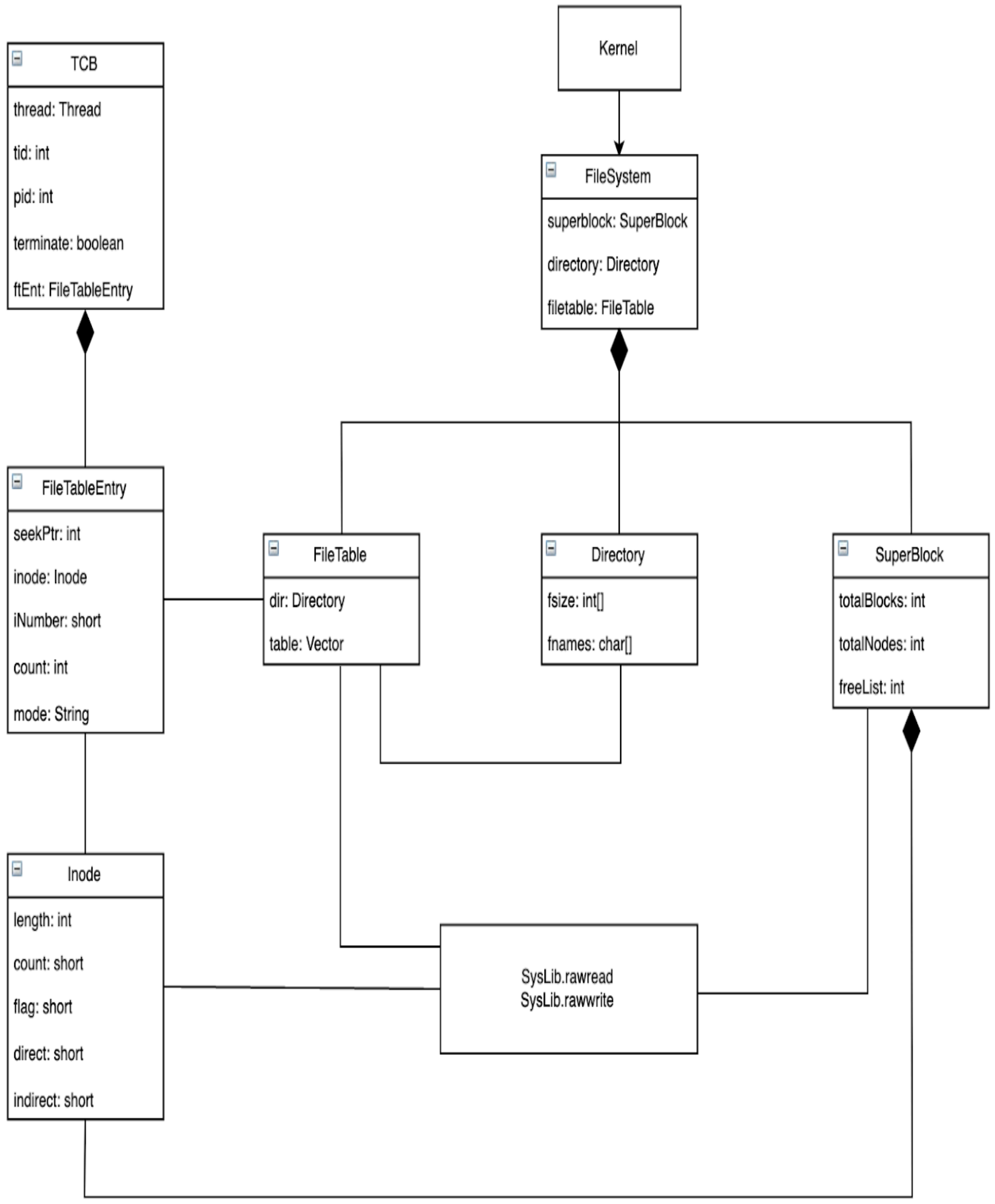
How to Compile

```
$javac [FileName.java]
```

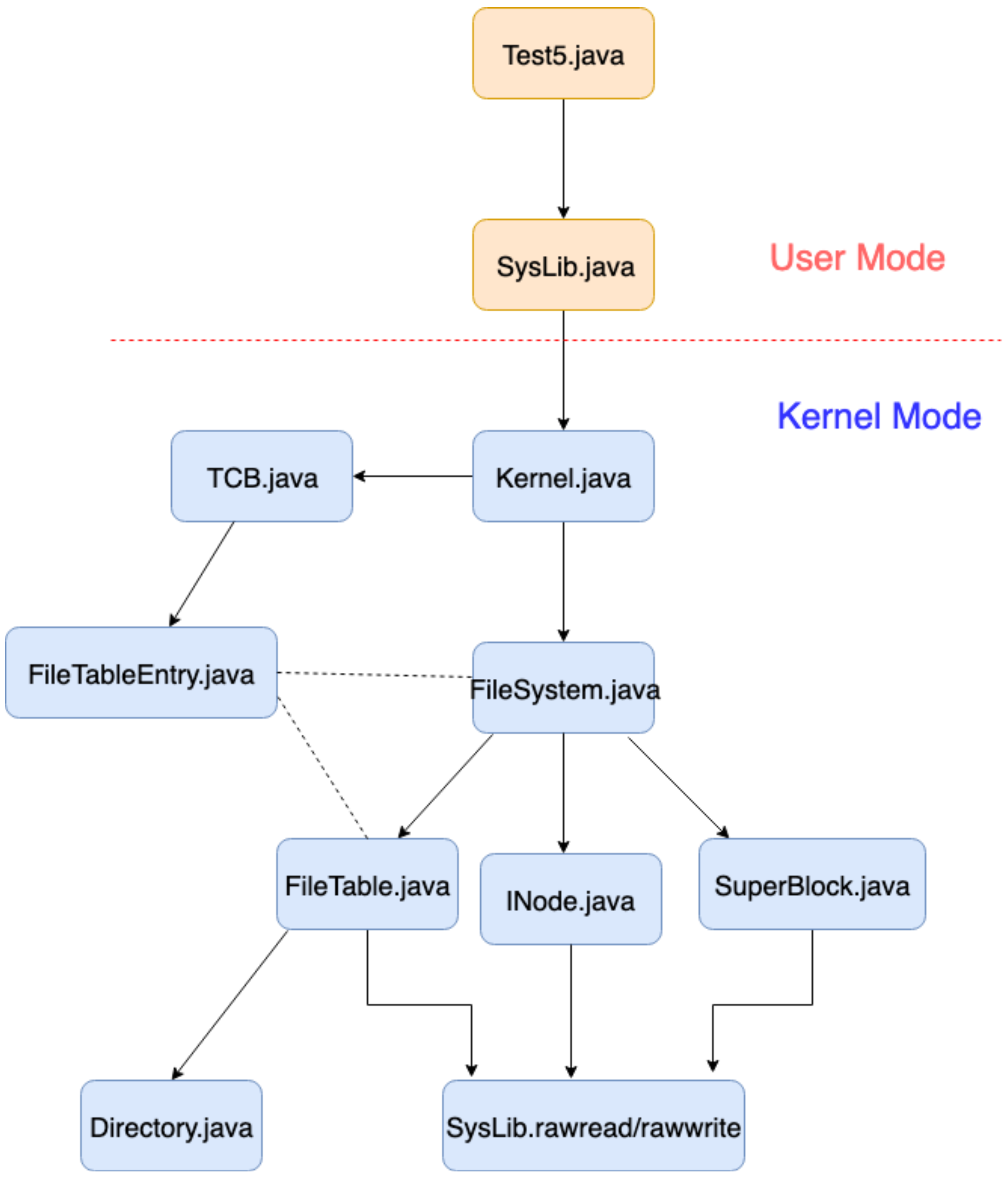
How to Run

```
$ java Boot
$ l Test5
```

Descriptions on Internal Design



Flow Chart of the File System



Explanation of Design

1. The SysLib will make calls to Kernel, and Kernel will instantiate the FileSystem and make FileSystem calls according to flags passed in by SysLib.
2. In FileSystem, the files can be opened, close, read, write, write+, append, and delete.
3. FileSystem holds instances of the SuperBlock, Directory, and FileTable objects
FileTable keep tracking of all currently opened files through a table of FileTableEntry
FileTableEntry is an object representation of an inode from the disk, thus holds an inodes information. If a file closes, the FileTableEntry object holds the updates made from a thread. Therefore, it copies it back to disk, and then removes the FileTableEntry from the table.
A FileTableEntry is a representation of a file descriptor for the TCB, thus holds information about modes, number of threads share this entry, and a seekPtr keeps track of where the thread left.
FileTable has a reference to the Directory to search for existing files, and keep tracking of free inodes.
    b. Directory keeps tracking of all existing files from the disk. This directory maps files to its iNode. Therefore, it also can be used as a representation of which inodes are in use of free to use from the disk.
    c. Superblock keeps tracking of free available blocks, allocates free blocks, deallocates used blocks and put it back in the pile of free blocks.
4. FileSystem, FileTable, Directory, and SuperBlock all use SysLib.rawread and SysLib.rawwrite to communicate and sync from the disk.

Considerations

Current Functionality

Current File System can support format, open, read, write, close, delete, fsize system calls. Users have different access modes for file system including read, write, write/read, and append. Each direct pointer is pointing for size 512 of block. The indirect pointer is pointing for size 256 direct pointers of index block.

Assumptions

- We assume that the disk can contain 1000 blocks.
- We assume that there are no other directories except the root will be provided by the system or created by the users dynamically.
- User does not directly access via the shell but provide disk commands through Java tests.

Limitations

- When it comes to write and read from the disk, we don’t use cache which means we need to use SysLib.rawread and SysLib.rawwrite. This takes longer process time than using cache.

- After the file system is initialized with the corresponding disk blocks and inodes, it is not possible to increment the inodes otherwise recreate and reformat the file system's properties.
- We don't have nested structure of the directory. In real life, root directory is on the root/top of the other subdirectories. However, in this project, we are assuming that all the files are located horizontally at the root level. No other directories are provided by the system and created by users dynamically.
- Lack of the permission or protection system for the file system. In the real life file system, there is access control to the file system to protect files and unauthorized access. Our methods are mostly public.

Performance Estimation

We use Test5 to test our program for several times. The test result shows that four file system successfully passes each test of system calls.

Possible Extended Functionality

- We may improve the project by integrating cache that we worked in P4. By doing that way, we may reduce the frequency of doing SysLib.rawread and SysLib.rawwrite and utilize the cache to improve the performance.
- We may improve the project by allowing users to grow the number of the inodes dynamically. Currently, after we decide on the number of creating inode, it cannot be changed. This need to involve other data structures to come in play and need more complex algorithms and memory.
- It can add the double index or triple index blocks to make the file size larger to store larger file.

Implementation

There are various java class files that are necessary to implement the file system on the ThreadOS: FileSystem.java, Superblock.java, Inode.java, Directory.java,
Modified files are: SysLib.java, Kernel.java, Scheduler.java

SysLib.java

read(int fd, byte[] buffer)
The function called from Test5.java and calls Kernel.java's READ case. It will allow File System to read the given file descriptor to the given byte buffer.

write(int fd, byte[] buffer)
The function called from Test5.java and calls Kernel.java's WRITE case. It will allow File System to write the given file descriptor to the given byte buffer.

open(String filename, String mode)
The function called from Test5.java and calls Kenel.java's OPEN case. It will accept parameters of filename and mode which would be stored in an argument array and pass through the Kernel interrupt.

close(int fd)
The function called from Test5.java and calls Kernel.java's CLOSE case. It will accept a file descriptor as a parameter and pass that to the Kernel which finds the File Table Entry and closes it.

fsize(int fd)
The function called from Test5.java and calls Kernel.java's SIZE case. It will accept a file descriptor as a parameter and pass that to the Kernel which finds the File Table Entry and finds the size of the file.

seek(int fd, int offset, int whence)
The function called from Test5.java and calls Kernel.java's SEEK case. It will accept a file descriptor, offset, whence as parameters. Then, it passes the file descriptor and arguments to find the corresponding file.

format(int files)
The function called from Test5.java and calls Kernel.java's FORMAT case. It will accept files as integer value and pass that to the Kernel which calls File System's format function to format.

delete(String fileName)
The function called from Test5.java and calls Kernel.java's DELETE case. It will accept a fileName as a parameter and pass that to the Kernel which calls File System's delete function to delete the file.

Kernel.java

Case READ
The READ case in Kernel.java gets current thread's TCB and finds the File Table Entry of the given file descriptor and calls File System's read function with the given File Table Entry.

Case WRITE
The WRITE case in Kernel.java gets the current thread's TCB and finds the File Table Entry of the given file descriptor. Then it calls File System's write function with the given File Table Entry.

Case OPEN
The OPEN case in Kernel.java gets the current thread's TCB and calls the File System's open function with the given fileName and the mode. It will return the current thread's file descriptor.

Case CLOSE
The CLOSE case in Kernel.java gets the current thread's TCB and gets the File Table Entry of the given file descriptor. It will call File System's close function to remove the file.

Case SIZE
The SIZE case in Kernel.java gets the current thread's TCB and File Table Entry. Then, it will call File System's fsize function to get the size of the file of the File Table Entry.

Case SEEK
The SEEK case in Kernel.java gets the current thread's TCB and File Table Entry with the offset and whence parameters. It calls File System's seek function to get the position of the file.

Case Format
The FORMAT case calls File System's format function with the given file descriptor.

Case Delete
The DELETE case calls File System's delete function with the given file name.

Directory.java

Directory(int maxInumber)
Constructor that initializes variables that are necessary for Directory class. There are two arrays used in Directory class that are fsizes, and fnames. Their length are initialized to maxInumber and fnames can contain maximum 30 characters for each index. The constructor Directory also initializes the root directory "/" with corresponding inode 0 in the disk block.

bytes2directory(byte data[])
The method which reads the given byte array from the disk. The parameter data[] contains information about the directory and the method initializes a directory instance with the given byte array. The meaningful information that should be retrieved from the data[] are file size and file name. The method utilizes SysLib.bytes2int method to get the fsize value from byte data and Java's String library converts byte info to the filename.

directory2bytes()
This method converts directory information into a byte array. The temporary byte array is created which can fit the all the meaningful information including fsizes array and fnames array. The method utilizes SysLib.int2bytes method to convert directory file size info to the bytes. Also, Java's String library converts file names to the byte data.

ialloc(String fileName)
The method allocates the slot for the given file name. It will find the empty slot in the fileSizes array with the value 0 and initialize it to the length of the given file name. Also, the given file name is stored as a sequential character in the fnames array. The method returns the allocated inode for the given file name.

ifree(short iNumber)
The method disallocates the directory entry for the given iNumber. It resets fsizes array to 0 for given inode number and remove the entry in the fnames array. Returns true if disallocation is successful, otherwise returns false.

namei(String fileName)
This method finds the inumber for given file name. It will loop through the fsizes array to see whether it contains the same length value with the given file name's length. If it finds one, it will compare the fNames array's value with the given file name whether they are matching each other. Returns found inode number otherwise, return -1.

SuperBlock.java

Superblock indicates the first block in the disk, block 0. It is used to describe the overall information about the file system including the information regarding the number of blocks, the number of inodes, the block number of the had block of the free list.

SuperBlock(int diskSize)
Constructor that reads the disk block 0 and converts the necessary information and store to the variables. For example, variable totalBlocks, totalInodes, freeList are found by calling SysLib.bytes2int function and stored into those variables.

sync()
This method write back current file system information back to the disk block 0. It will call SysLib.int2bytes function to convert the integer variables into the bytes, store in the temporary blockInfo array and write to the disk.

getFreeBlock()
This method deques and returns the first free block from the free list. The method create a temporary byte array and call SysLib.rawread to get the block information of the first free block. Then, updates freeList variable to the next available free block. Returns free block number information.
returnBlock(int blockNumber)
This method enques the given block to the front of the free list. The method create a temporary byte array and call SysLib.rawread to get the free block information of the current free block. Then it call SysLib.rawwrite to save that info to the given block number and updates the free list to the given block number.

format(int numberOfBlocks)
This method resets total inodes and free list based on the given numberOfBlocks. The method creates a new Inode for the number of given blocks as a parameter. Then, it links free list blocks each other by calling SysLib.int2bytes to get the next free block number information and calling SysLib.rawwrite to store that info to the current available block.

Inode.java
Inode()
Constructor that initializes variables that describes an INode. INode is used to describe a file in the file system. The variables that describe an INode include length, count, flag, direct and indirect pointers. The constructor sets those variables as an initial state.

Inode(short iNumber)
Constructor that retrieves the existing inode from the disk into the memory. The method creates a temporary byte data array and calls SysLib.rawread to put the block information of corresponding INode to the array. Then it uses SysLib.bytes2int, SysLib.bytes2short functions to get length, count, flag, direct and indirect pointer values for the retrieved inode and initializes a new INode with those values.

toDisk(short iNumber)
Saves the inode information to the disk. The method creates a temporary byte data array and uses SysLib.int2bytes, SysLib.short2bytes functions to convert integer and short values of variable to the byte value. The method calculates the block number for corresponding INode number. Then, it calls SysLib.rawread to read in the existing block that contains INodes and replaces with the new INode information with the correct offset. Finally, it calls SysLib.rawwrite to write back the new block information to the disk.

findIndexBlock()
This method find the index block number by returning the value of indirect pointer.

registerIndexBlock(short blockNumber)
This method registers the given block number as the index block. It sets indirect pointer value to the given blockNumber as a parameter. Then, it creates a new index block with 256 size and set it as a initial state and call SysLib.rawwrite to write into the disk.

findTargetBlock(int offset)
This method finds the target block with the given offset value. If target block index is less than 11, the target block is found by direct pointer. Otherwise, it can be find in the indirect pointer by reading in the block that is pointed by the indirect block using SysLib.rawread.

registerTargetBlock(int offset, short block)
This method registers the block at the given offset and the block number. For the direct pointers, If the targeted block is already occupied the method throws the error. Otherwise, it will register the block with the target position. For the indirect pointers, the logic applies the same. If the targeted block is already occupied, throws an error, otherwise update the indirect pointer block.

unregisterIndexBlock()
This method unregisters index block by resetting indirect pointer to -1. It will call SysLib.rawread to read the what had been stored in the indirect pointer block and return it with the successful unregister.

FileTable.java

FileTable(Directory directory)
This is a constructor which initializes a new file table and directory of the file system.

falloc(String fileName, String mode)
This method allocates a new file table entry for the given file name and the mode. The method looks for the inumber for given file name in the directory. If it couldn't find the inumber and the mode is read, it returns null. If it finds the inumber in the directory and it is meets file access condition, it will increase inode counter, write back the new inode info to the disk with the inumber, creates a new entry with given inode, inumber, mode, and add the entry to the table.

ffree(FileTableEntry entry)
This method is remove the data from file table entry. use boolean to check the data is in file table entry or not. If it is true, it need to remove it. if it is false, it set invalid and cannot find. We set the reference from file table. Check the table for read and write data and remove it,. Then, decreasement for count the users number. Otherwise, return false.

fempty()
This method is check the file table is empty or not. If the file table is empty, it will be return to call the stating format.

FileSystem.java

FileSystem()

This is a constructor which initializes variables that are used for the File System. The SuperBlock, Directory, File Table are initialized and reads in a root directory with the name "/" and "r" mode.

format(int files)
This methods formats the content of the disk based on the given number of files. The method calls format function of the superblock which will return with the correct number of inodes and the free list. It will create the directory based on the number of the number of inodes of the superblock and create the file table of that directory.

close(FileTableEntry entry)
This method closes the entry from the file table of the file system. If the entry is null, it will return false. We need to synchronize the entry to prevent modifying/accessing from other user threads. The method decrement the counter of the entry and if that count reaches 0, it will remove the entry from the file table.

delete(String filename)
This method deletes the file from the file system. We creates the File Table Entry by call open function with given file name with write mode. Then we can update the directory and remove the entry from the file system by accessing iNumber field. Return true if successful, otherwise return false.

fsize(FileTableEntry entry)
This method returns the size of the file of the given File Table Entry. We need to synchronize since in the meantime we are getting the file size, it might possible that some other thread can modify the size.

deallocAllBlocks(FileTableEntry entry)
This method deallocates all blocks for the corresponding File Table Entry. The method will loop through direct pointers of the File Table Entry's inode to see whether there are valid blocks. Then it calls returnBlock function in SuperBlock.java to return the block to the free list. Also, it will look into it to see whether indirect pointer contains valid block. It will deallocate and put it back to the free list as well.

seek(FileTableEntry entry, int offset, int location)
This method updates the seek pointer of the File Table Entry. If the user attempts to set the seek pointer to a negative number, it should be set to 0 which is the start point of the file. If the user attempts to set the pointer to beyond the file size, the seek pointer must be set to the end of the file.

- If location == SEEK_SET, the file's seek pointer is set to offset bytes from the beginning of the file
- If location == SEEK_CUR, the file's seek pointer is set to its current value plus the offset. The offset can be positive or negative.
- If location == SEEK_END, the file's seek pointer is set to the size of the file plus the offset. The offset can be positive or negative.

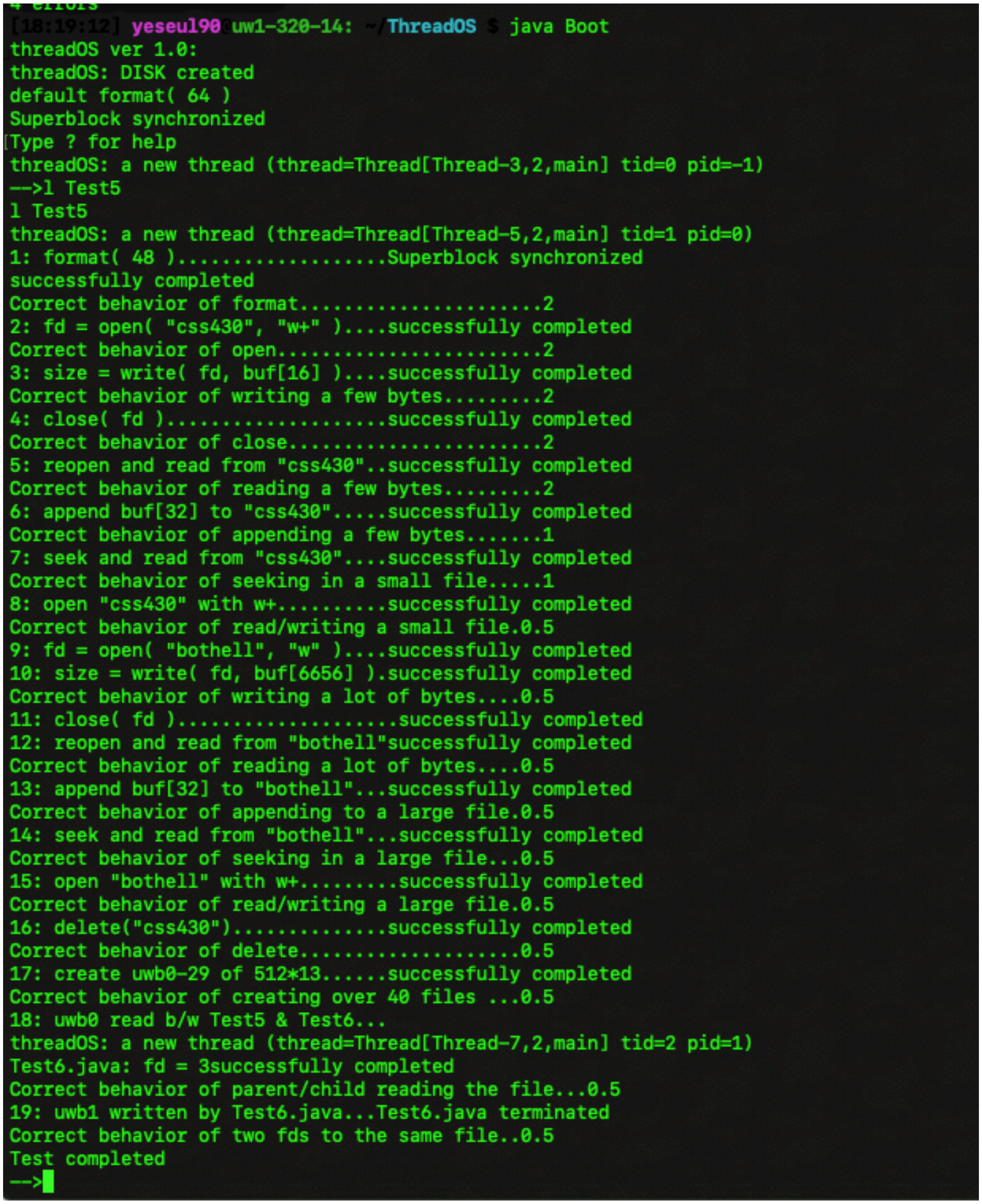
open(String filename, String mode)
This method opens the file with the given fileName and the mode. It will create the File Table Entry with falloc method. If the mode is write, it will deallocate blocks of corresponding File Table Entry's inode in the current file table. Otherwise, return the File Table Entry.

read(FileTableEntry entry, byte[] buffer)
This method reads the file up to the length of the buffer. This method keeps track of the offset, the blocks left, file left, and the seek pointer in the while loop and copies data to the buffer as long as the buffer is filled up. The File Table Entry has to be synchronized.

write(FileTableEntry entry, byte[] buffer)
This method writes the content from the buffer to the File Table Entry starting at the position indicated by the seek pointer. If the File Table Entry's mode is read which means someone is reading the file at the same time, it needs to return -1. The method calls the findTargetBlock function from Inode.java in order to find the target block to write. If it couldn't find the target, it needs to create the new block from calling getFreeBlock function from SuperBlock.java. Then it registers the new block for the entry. For the retrieved or created block, the method writes the data in the buffer to the block and saves back to the disk.

sync()
This method writes the data back to the disk by calling directory2bytes function in Directory.java and sync function in SuperBlock.java

Test Results



```
4 errors
[18:19:12] yeseul90@uw1-320-14: ~/ThreadOS $ java Boot
threadOS ver 1.0:
threadOS: DISK created
default format( 64 )
Superblock synchronized
[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 )...................Superblock synchronized
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
-->
```