Program 5: File System Report

Results

Test5.java with new DISK creation:

```
lisakim@cj:~/Desktop/CSS430_FileSystem/src $ java Boot
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
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
```

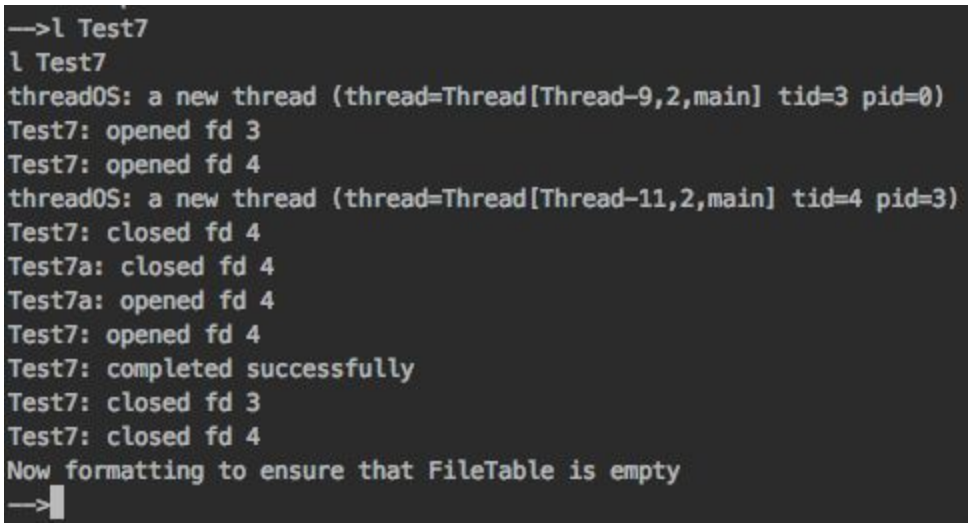
```
10: size = write( fd, buf[6656] ).successfully completed
Correct behavior of writing a lot of bytes....0.5
11: close( fd ).....successfully completed
12: reopen and read from "bothell"successfully completed
Correct behavior of reading a lot of bytes....0.5
13: append buf[32] to "bothell"...successfully 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
-->
```

Test5.java with existing DISK, with different diskBlock 80:

```
lisakim@cj:~/Desktop/CSS430_FileSystem/src $ java Boot
threadOS ver 1.0:
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Test5 80
l Test5 80
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
1: format( 80 ).....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
-->
```

Test7.java, after test results above:



File Specifications

SuperBlock

SuperBlock is the class that formats the disk. Responsible for allocating and deallocating blocks, setting total blocks and the available inodes for each block, and keeping track of current free blocks available at all times.

Methods:

Superblock(int diskSize): constructor that takes in the diskSize to set the disk information as a whole

format(int inodeBlocks): Reformats the disk by setting total inodes and free list. Intializes each inode blocks to an instance of inode. Intializes each free block with 512 byte array. Links the free blocks together

sync(): syncs total blocks, inode blocks, and free linked blocks back to super block 0

getFreeBlock(): returns the free block available by sets the free list to the next free block

returnFreeBlock(int blockNumber): frees the passed in block by wiping out the block to 0's, set this block to point to the current freelist, and set this block as the current free block

Inode

Each file in the file system will be represented by an inode. Each inode contains 32 byte size of: file size, flag to indicate if it's in unused/use/read/write, number of threads that are using this file currently, total of 11 direct pointers and 1 indirect pointer that points to blocks of Data.

Methods:

Inode(): default constructor that creates blank inodes

Inode(short iNumber): constructor that takes in an inode number, grabs this inode from the disk, and sets all instance variable with this data from the disk

toDisk(short iNumber): a write-back operation that saves this inode information to the inode in the disk

getIndexBlockNumber(): returns the indirect pointer

setIndexBlock(short indexBlockNumber): receives a free block to format it to 256 indirect pointers and sets this free block number as the indirect pointer

findTargetBlock(int offset): finds and returns the target block where the offset is located

setDirectPointers(short blockNumber): sets the free block to a free direct pointer

setIndirectPointers(short blockNumber): sets the free block to a free indirect pointer

FileTableEntry

File table entry is actually a object presentation of an inode (file) in the memory. Each file descriptor should correspond to exactly ONE of FileTableEntry because each file descriptor can open its file in its individual mode. It contains seekPtr, inode, iNumber, count of the threads that are working on this file entry and mode of the operation.

Methods:

FileTableEntry(Inode i, short inumber, String m): constructor that sets the passed in arguments to the corresponding instance variables

FileTable

Keeps track of files (FileTableEntries) that are currently opened. It allocates a new FileTableEntry for new files to the table, inserts the new file into the directory and removes it from the FTE when a thread is finished using the file (close file, not delete)

Methods:

FileTable(Directory directory): constructor that receives a reference to the directory created in FileSystem, initialize a vector of FileTableEntry

falloc(String filename, String mode): allocates a new file (if it does not exist in the directory), and creates a FTE and inserts it to the FTE table and directory

ffree(FileTableEntry entry): writes back updates to inode to the disk, removes the file from the FTE table

fempty(): tells if a FTE table is empty or not

Directory

Directory is the file that keeps track of all the files on the disk with their respective names. It contains two arrays, one to hold each file name size and the other one is a two dimensional array that contains files name. First row in both holds the root of the directory "/". Array index corresponds to iNumber, thus can be also used to keep track of available inodes.

Methods:

Directory(int maxInumber): constructor that makes intializes that creates a directory that holds maxInumber files

bytes2directory(byte data[]): takes the data from disk and intializes the directory instance with this given data

directory2bytes(): converts the entire directory into bytes and writes it back to disk

ialloc(String filename): "allocates" a new "inode" by setting the filename into the next available space in the directory

ifree(short iNumber): "deallocates" this inode and deletes it from the directory

namei(String filename): given the filename, search for it in the directory

TCB

TCB had to be updated. Both initial file and the updates needed were given to us. No changes made from us.

FileSystem

FileSystem is in control of actions including open, close, read, write, append, seek, delete and format. It has access to FileTable, Directory and superblock and through FileTable it can grab each FileTableEntry and its inode.
It also handle sets the seekPtr based on each mode:

- **r, w and w+ sets the seekPtr to beginning of the file.**
- **a will set the seekPtr to the end of file.**

Methods:

format(int files): will call superblock format if the file table is empty.

open(String filename, String mode): allocates a file to filetable from disk with the respective mode (r, w, w+, a)

close(FileTableEntry ftEnt): close an open thread related to ftEnt from fileTable and sets all the counts for the ftEnt and related indoe and set the flag in inode.

fsize(FileTableEntry ftEnt): returns size of the file

read(FileTableEntry ftEnt, byte[] buffer): reads the data of ftEnt from disk and stores it into buffer

write(FileTableEntry ftEnt, byte[] buffer): writes and/or append the data from buffer into the location of the ftEnt file

deallocAllBlocks(FileTableEntry ftEnt): deallocate and free all the used blocks and write them back to disk as a free block.

delete(String fileName): deletes the file from the disk specified by filename. All blocks used by file will be freed.

seek(FileTableEntry ftEnt, int offset, int whence): move the seek pointer by offset amount based on whence:

- **SEEK_SET: seek pointer will be set to offset from beginning of file**
- **SEEK_CUR: seek pointer will be set to current position + offset**
- **SEEK_END: seek pointer will be set to end of file + offset (offset can be negative)**

SysLib

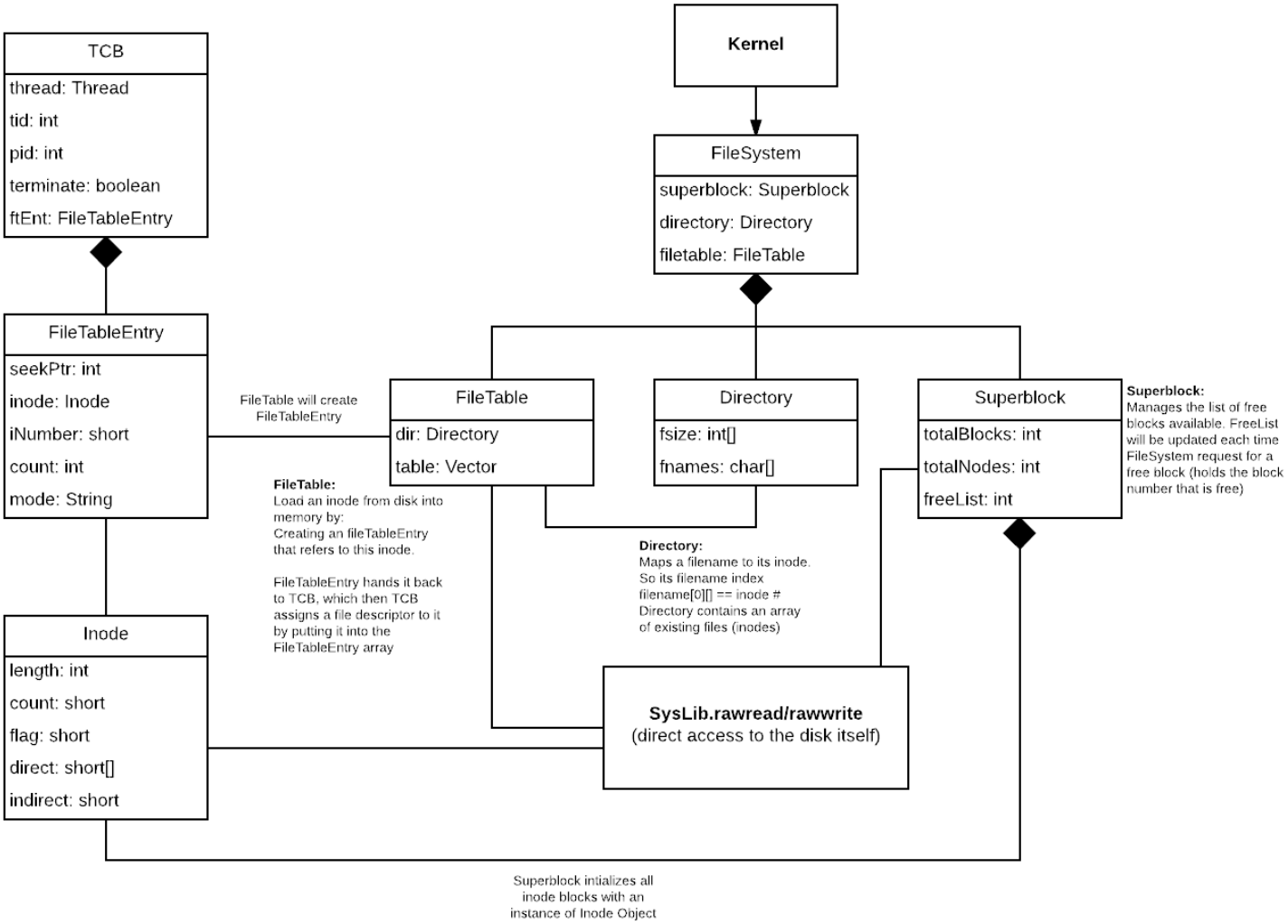
SysLib is the file that calls most functions for kernel. We added missing interrupt calls needed to run our file system such as format, open, close, write, read and delete.

Kernel

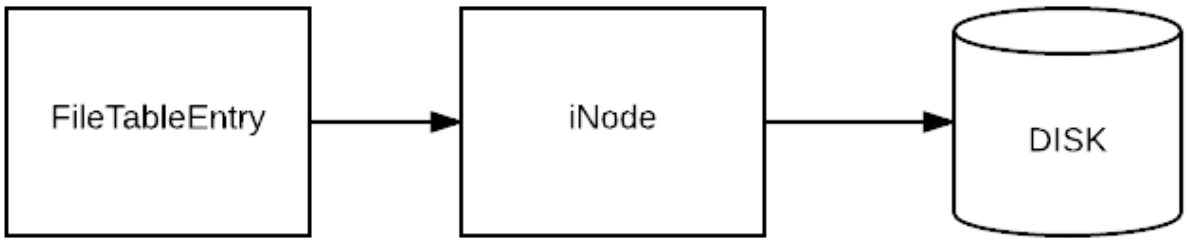
Inside Kernel we had to add different cases for the given flag by SysLib. All we had to do was to check for the edge cases and call the related function to the current flag from FileSystem.

Descriptions of Design

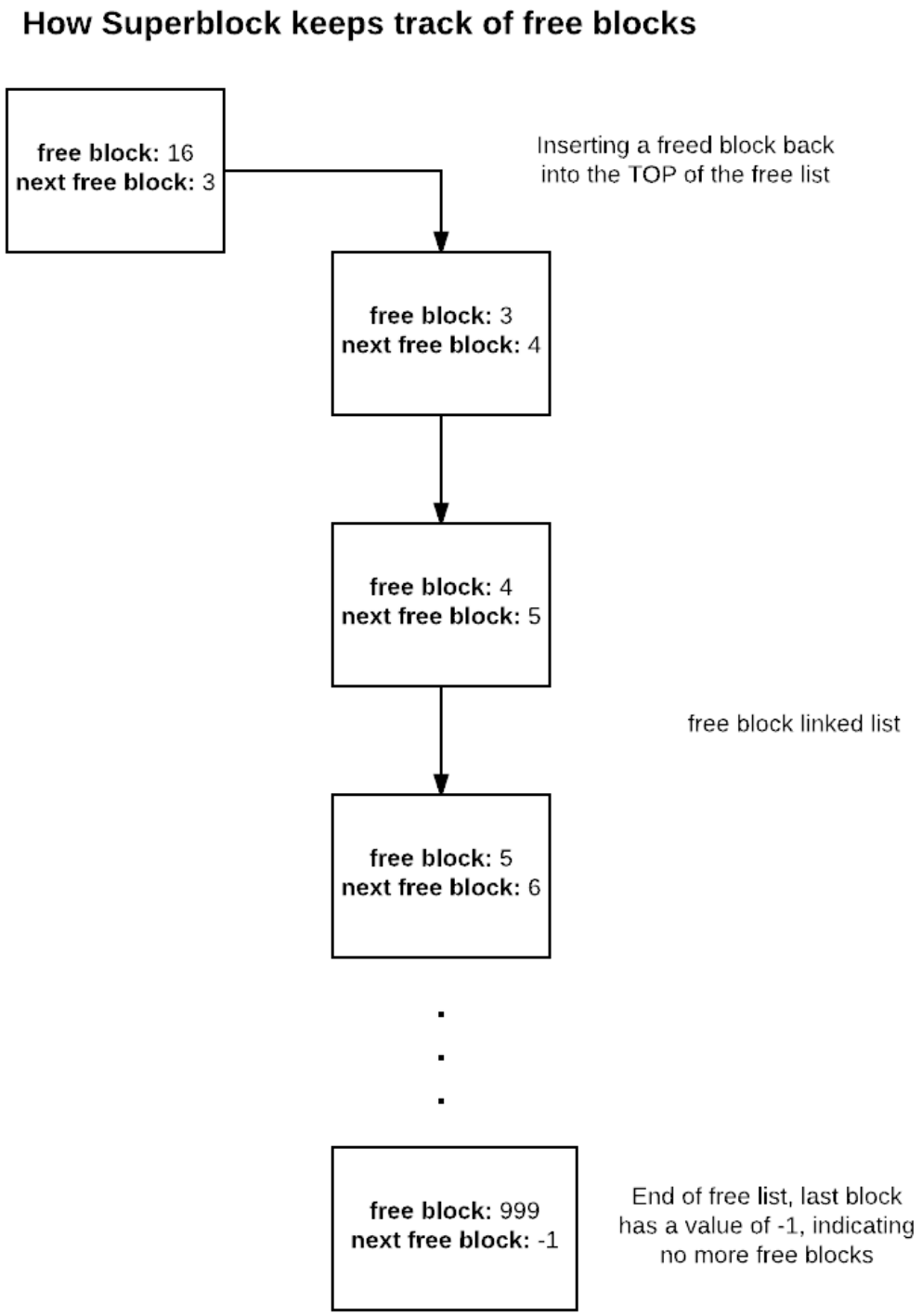
Overview of Program



How a file is represented in our FileSystem



FileTableEntry is a object representation of an iNode from memory



Explanation of Design:

1. **From SysLib, it will make calls to Kernel, and Kernel will insantiate the FileSystem and make FS calls according to flags passed in by SysLib**
2. **In our FileSystem is where files can be opened, read, write, write+, append, close, delete.**
3. **FileSystem holds instances of the FileTable, Directory, and Superblock objects**
    a. **FileTable keeps track of all currently opened files through a table of FileTableEntry**

i. **FileTableEntry is an object representation of an inode from the disk, thus holds an inodes information. If a file closes, the FTE object holds the updates made from a thread, so it copies it back to disk, and then removes the FTE from the table.**
    1. **A FTE is a representation of a file descriptor for the TCB, thus holds information about modes, # threads sharing this entry, and a seekPtr that keeps track of where the thread left of.**
ii. **FileTable also has a reference to the Directory to search for existing files, and keeping track of free inodes**

b. **Directory keeps track of all existing files from the disk. This directory maps files to its iNode, thus is also can be used as a representation of which inodes are in use or free to use from the disk.**
c. **Superblock keeps track 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 uses SysLib.rawread and SysLib.rawwrite to communicate and sync to and from the disk**

Considerations

Performance Estimation:

Our current program passed both Test5 and Test7. For thorough testing, we ran Test5 and Test7 concurrently several times. We also tested with DISK creation, and with existing DISK. Based on these tests, we believe that we achieved a pretty good accuracy.

As for the speed of running our program, there are some delays when calling the SysLib.format() function as we expected. Formatting re-configures and re-initializes the disk to whatever the value of total blocks were sent in.

Current Functionality:

Current FileSystem can handle and store files up to 136704 for each file. This is happening inside the file system by using each inodes 11 direct pointers and 1 indirect pointer.

Each direct pointer is pointing to a block of size 512. Indirect pointer is pointing to an index block that holds 256 direct pointers.

Current FileSystem has the ability to format and creates a blank disk data. It also is able to write read and append to a file.

Possible Extended Functionality:

One possible extended functionality can be adding double index or triple index blocks to increase the file size that we can store.

Adding a GUI can be another Extended Functionality for the current fileSystem. A simple to use GUI can enhance the user experience.