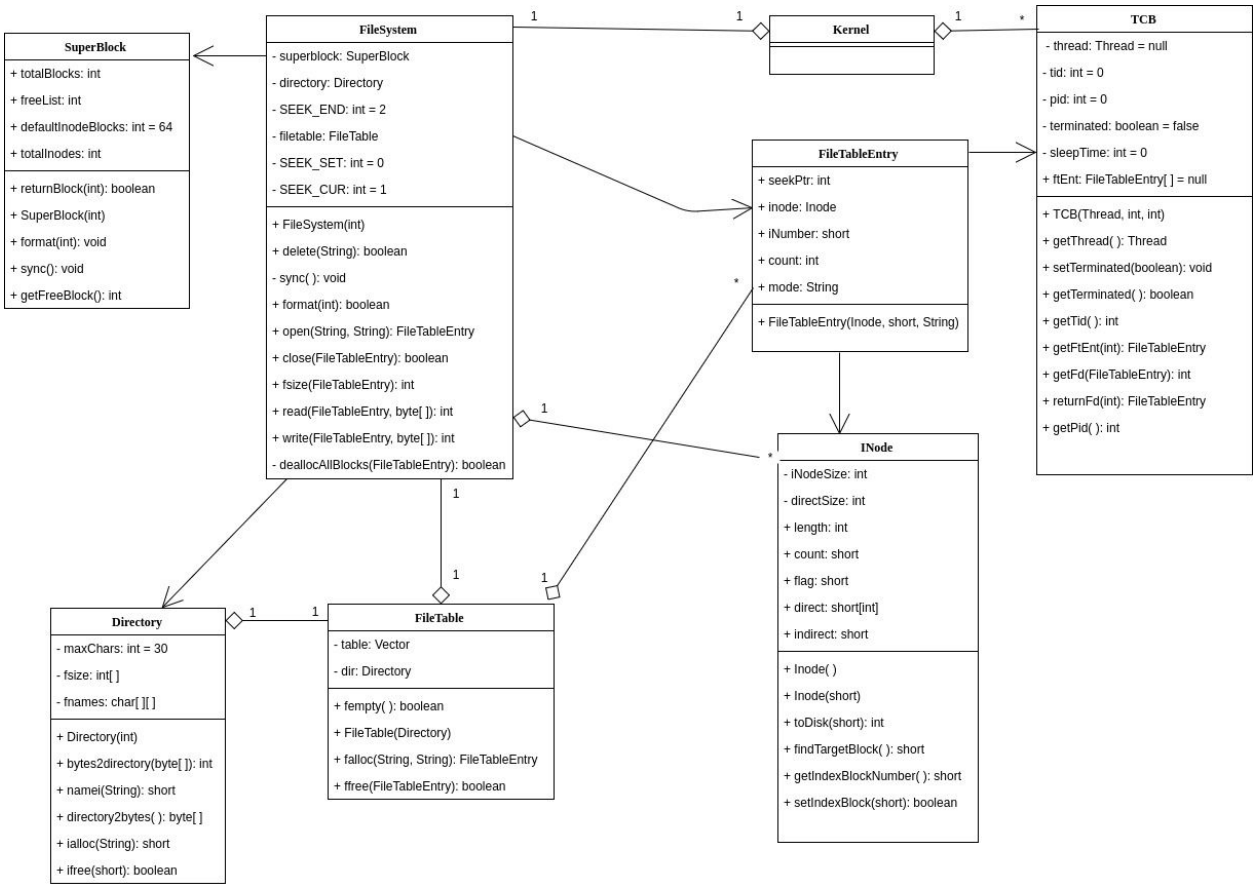
Edward(Seungho Chang)
Bardia Borhani
Sandun Ranaweera

Final Project - File System

Design

Part 1: Class Relationship



Part 2: Class Description

Directory - Directory.java

The purpose of the directory is to map filenames to inodes.

Fields:

- **Int fsize[]:** is an int array that holds the size of the names of all files in the directory. For example, the first index holds the size of the root name which is ”/” - so fsize[0] holds 1 as “/” has a length of 1.
- **Char fnames[][]:** holds all the names of the files as a character array. Integers are used to index the 2-d array. Each column will contain a name of a file as a character array. Fnames[0] will return [‘/’].

Methods:

- **void bytes2directory(byte[] data):** intakes a byte array that contains directory information retrieved from the disk. The information is extracted from the byte array and is placed into the class’ fsize[] and fnames[][] variables.
- **byte[] directory2bytes():** Does the opposite of what bytes2directory does. It places all the values of the dictionary class into a byte array and is returned.
- **Short ialloc(String filename):** intakes a string representing a file name. Allocates a new inode number for this filename
- **boolean ifree(short iNumber):** deallocates this iNumber (inode number). The corresponding file will be deleted.
- **short namei(String filename):** namei intakes a string representing a file name. The inode associated with the filename is returned.

Inode - Inode.java

The basic data structure for **File System,** keeping track of which blocks are in the file(and their order). A inode object is created in the FileTable.java class under the falloc method - then a new FileTableEntry object is made and is assigned a reference to the newly created inode object

Fields:

Inode has two size variables**; inodeSize** and **directSize**

- **inodeSize:** represent the size of **Inode** Object**,** fixed to have 32 bytes data of file details
- **directSize**: the number of direct pointers

Methods:

Inode can be **retrieved from disk** by the number of Inode or **saved to disk** as the i-th Inode.

- **Int toDisk(short iNumber):** saved to disk as the i-th **Inode**
- **Short getIndexBlockNumber():** return the index block number

- **Boolean setIndexBlock(short indexBlockNumber):** set the index block number
- **Short findTargetBlock(int offset):** Find the target block using the offset variable and return it

Superblock - Superblock.java

OS managed structure used to describe the followings:

1. **The total number** of disk blocks
2. The number of **Inodes**
3. The block number of the head block of **the free list**(the list of empty blocks)

Fields:

- **totalBlocks:** the number of disk blocks
- **totalInodes:** the number of **Inodes** in this whole block
- **freeList: :** the block number of the head block of the empty bolck list's head

FileTable - FileTable.java

System-wide file table that keep tracks of every file usage in the system. It contains **FileTableEntry**.

Fields:

- **Vector table:** This vector contains FileTableEntry objects (description about FileTableEntry class is below).
- **Directory dir:** A variable of type Directory is created so that when this class creates inodes, it can inform the directoy an inode is created or deleted

Methods:

- **FileTableEntry falloc(String filename , String mode):** Intakes a parameter representing a file name and another parameter indicating how the file is wished to be accessed: r = read, write = write, a = append, etc. A new Inode and FileTableEntry object is created and the FileTableEntry references the newly created Inode object
- **boolean ffree(FileTableEntry e):** Receives a fileTableEntry reference, saves the corresponding inode to the disk and frees this fileTableEntry object
- **boolean fepty:** Checks if the field "table" is empty

FileTableEntry - FileTableEntry.java

A **FileTableEntry** is a data structure used as an entry object for referencing to an **Inode**.

Fields:

- **seekPtr:** a file seek pointer to know the location within a file
- **inode:** a reference to its inode
- **iNumber: inode** number
- **count:** the number of thread sharing this entry
- **mode:** represents the security level of file with “r”, “w”, “w+”, or “a”

TCB - TCB.java

ThreadControlBlock maps between file descriptor numbers and **FileTableEntry**

Fields:

- **thread:** the current thread that uses **FileSystem**
- **tid:** id of **thread**
- **pid:** id of parentTid
- **terminated:** boolean value for termination status
- **ftEnt: FileTableEntry** object for mapping

Methods:

- **Getters and setters for each field**

FileSystem - FileSystem.java

FileSystem maintains disk space in blocks and allocates available blocks to each stream-oriented file with the data structures of **Directory, FileTable** and **FileTableEntry.** Creates **SuperBlock** which maintains information about disk blocks.

Fields:

- **superblock:** the data structure at **block#0**
- **Directory: Directory**-structure cache holds directory of recent dir access
- **fileTable:** contains a copy of the **Inode** of each open file and other information

Methods:

- **int SysLib.format(int files):** Formats the disk by deleting the contents of it
- **int SysLib.open(String fileName, String mode):** Opens the file specified by the *fileName* string in the given *mode* (where "r" = ready only, "w" = write only, "w+" = read/write, "a" = append)
- **int read(int fd, byte buffer[]):** reads into the buffer[ ] array the contents of a file specified by the file descriptor value (fd)
- **int write(int fd, byte buffer[]):** writes the buffer[ ] array contents to the file specified by the file descriptor value (fd)

- **int seek(int fd, int offset, int whence):** Updates the seek pointer of a given file pointed to by the file descriptor. The pointer is moved by the value of “whence”
- **int close(int fd):** Closes a file
- **int delete(String fileName):** Deletes a file
- **int fsize(int fd):** return the size of each file