Cairo University

Faculty of Computers & Information

Advanced Operating System (2021-2022)

Assignment 3

Virtual File System

Submission Instructions

- This assignment will be submitted into groups of (3-4 students) and the due date is on 22/5/2022 at 11:59 PM on Blackboard.
- You should work in groups from the same lab.
- You can deliver code in (Java, C++, C#, or Python). You should deliver a running code and match the assignment criteria.
- You should submit your code on Blackboard before the specified time. Follow these steps in your submissions or you will lose your grades (No Excuses)
- You should deliver your own code. Don't copy from the other teams or the internet **(NO CHEATING)**. Otherwise, you will get a negative grade.
- Late submission is NOT allowed.
- Only one member from each group should upload a zipped file. Make the name of this zipped file <student_id1><student_id2><student_id3><student_id4>.zip. This zipped file must contain:
  - The Source Code
  - The **V**irtual **F**ile **S**ystem file
  - Text File named "Group.txt" which contains the group members names and ids

Overview Statement:

Assume that you have a virtual file system with a root directory called "root" all the files and folders will be stored under it. The disk size consists of **N** blocks and each block size is **1 KB**.

The aim of this assignment is to simulate the allocation and de-allocation of files and folders using different allocation techniques. Implement the two allocation techniques listed below (Refer to the file system chapter in your textbook):

1- Contiguous Allocation (Using Best Fit allocation)

2- Indexed Allocation

3- Linked Allocation

After running the application, the user will interact with your virtual file system through a series of commands, these commands are illustrated in the table below:

System Commands:

Command	**Summary**
`CreateFile root/file.txt 100`	This command used to create file named “file.txt” with 100 KB size under the path “root” Pre-requests: 1- The path is already exist 2- No file with the same name is already created under this path 3- Enough space exists
`CreateFolder root/folder1`	This command is used to create a new folder named “folder1” under the path “root” Pre-requests: 1- The path is already exist 2- No folder with the same name is already created under this path.
`DeleteFile root/folder1/file.txt`	This command used to delete file named “file.txt” form the path "root/folder1". Any blocks allocated by

	this file should be de-allocated. Pre-requests: 1- The file is already exist under the path specified
`DeleteFolder root/folder1`	This command used to delete folder named “folder1” form the path "root". All files and subdirectories of this folder will also be deleted. Pre-requests: 1- The folder is already exist under the path specified
`DisplayDiskStatus`	This command used to display the status of your Driver the status should contain the following information: 1- Empty space 2- Allocated space 3- Empty Blocks in the Disk 4- Allocated Blocks in the Disk
`DisplayDiskStructure`	This command will display the files and folders in your system file in a tree structure

Saving and loading Virtual File System

In this program we are not creating actual files and folder, we will just simulate having a series of blocks and these blocks will be allocated to files when created and will be de-allocated when these files are deleted.

Your virtual file system information like (the files information, the folders information, the allocated blocks and so on) should be saved on a file on your hard disk to be able to load it the next time you run the application.

So when the application starts, the system should automatically load the disk structure form the **V**irtual **F**ile **S**ystem file say named "c:\DiskStructure.vfs". Then the user will start to enter commands which will be executed on the loaded data in memory, and before the application terminates, the data in memory will be written into the file again.

Note: It's your task to choose which structure will be used to store this file

The following information should be stored in the file:

1- Files and Folders Directory Structure.

2- The Empty blocks of the virtual DISK

3- The allocated blocks in your virtual DISK and which files/folders are take these places.

More details about these 3 points are in the following sections.

1- Files and Folders Directory Structure

The **V**irtual **F**ile **S**ystem file should contain the structure of the files and folders which should be loaded in a tree structure in memory. This is required to be able to print the tree structure when the user requests the command "`DisplayDiskStructure`".

Example directory structure:

- **<root>**
  - **File0.txt**
  - **<folder1>**
    - **File1.txt**
    - **<folder2>**

You should determine how to save this structure in the **V**irtual **F**ile **S**ystem file and how to load it into Java Data Structure.

Programming Hint:

Note: You can use this hint if it helps you, but if doesn't help, you are free to leave it and make your own design.

You can make a class called "Directory" and a class called "File", and then the Directory class should contain a list of File type and another list of Directory type. The Files list will contain the files in this Directory and the Directories list will contain the subdirectories of this Directory.

The allocation technique simulator will have the root directory object only, and by this object it can navigate in the directory structure from the top to bottom using recursion.

For example to print the directory structure, we can make a function in the "Directory" class called `printDirectoryStructure(int level)` which prints the directory name and its files then makes recursion on the sub directories in the next level to print their directory structure too.

Code for illustration:

```
The allocation technique will have the root Directory object:
Directory rootDirectory;

And to print the directory structure:
rootDirectory.printDirectoryStructure(0);

class File {
	private String filePath;
	private int[] allocatedBlocks;
	private boolean deleted;
	. . .
}
public class Directory {
	private String directoryPath;
	private File[] files;
	private Directory[] subDirectories;
	private boolean deleted = false;
	. . .
	public void printDirectoryStructure(int level) {
		/*this method prints the directory name and its files
		then makes recursion to loop on the subDirectories to print their
		structure too.

		The level parameter can be used to print spaces before the
		directory name is printed to show its level in the structure */
		. . .
	}
}
```

2- Empty Blocks (Free Space Manager)

You should implement a free space manager component which will be used by the allocation technique to know which blocks are free, to search for free contiguous blocks, to de-allocate blocks when a file is deleted and to allocate blocks when a file is created and so on.

As mentioned before in the previous section, your virtual file system file will contain the empty blocks, this can be done using a series of zeros of ones, where a zero at index 2 in the series means that block 2 is free, and a one at index 2 means that block 2 is allocated.

Example, if you have 10 blocks where the first 5 are allocated and the last 5 are free, the series that should be saved in the file system file should be: `1111100000`

3- The allocated blocks for files

The needed information to represent the allocated blocks by files depends on the allocation technique, for example:

In the **contiguous allocation** you should store the start block number and the number of blocks a file is allocating like:

```
root/folder1/file.txt   0  5

root/folder2/test.txt   5  3
```

This means that the file root/folder1/file.txt is allocating 5 blocks starting at block number 0. And the file root/folder2/test.txt is allocating 3 blocks starting at block number 5.

In the **linked allocation**, you should store the start block number and the end block number for a file and the links between the blocks like:

```
root/folder1/file.txt   12  20

12   15

15   20

20   nil
```

This means that the file root/folder1/file.txt starts at block number 12 and ends at block number 20. After block 12, it's allocating 15, after 15 it's allocating 20 and after 20 is the End Of File.

In the **indexed allocation**, a block called the index block is allocated for each file to contain that file blocks numbers. So you should save the index block number for each file and the content of the index block like:

```
root/folder1/file.txt   7

7    1 2 3 4
```

This means that the index block of the file root/folder1/file.txt is 7. And the content of the block 7 is 1 2 3 4, which means that this file is allocating the blocks 1 2 3 and 4.

Again, you can use any format to represent this information in the **V**irtual **F**ile **S**ystem file, I'm just giving examples for illustration.