

DB101 - DATA MANAGEMENT

Fall 2012

Data Structures Project

GENERAL PROJECT INFORMATION

Course Name	DB-101 Data Management
Project Name	Virtual File System (VFS)
Project Description	VFS is logical view of the filesystem at a higher level of abstraction. The project supports basic operation for file management.
Project Type	Individual
Version	1.4
Document last updated at	17-Oct-12

PROJECT DESCRIPTION

A **virtual file system (VFS)** is an abstraction layer on top of a concrete file system that provides the ability manage information about files and folders present in the hard disk. The purpose of a VFS is to allow client applications to access different types of concrete file systems in a uniform way. A VFS can, for example, be used to access local and network storage devices transparently without the client application noticing the difference.

(Refer: http://en.wikipedia.org/wiki/Virtual_file_system)

All operations, file system operations are Unix type.

Glossary of Terms

- File System – A system that provides a unified mechanism for managing data that is organized in the form of files
- File Blocks – Fundamental storage unit into which files are actually stored. A single file may span across multiple file blocks
- File Descriptors – A data structure that contains meta-data information about the file (name, type, location, etc.)

Overview:

File System is basically a large file which stores the data contained in all the existing files. Each file/folder in a file system has a corresponding File descriptor which stores all the information of the file /folder e.g. Size, name, path etc. File system is divided into data blocks of fixed size which store the actual file contents. The files when saved in the file system can be scattered over multiple blocks depending upon the size of the file.

Example:

Suppose there is a file system of 10 Kb divided into 10 data blocks(0-9) of 1Kb each. Suppose there are three files a.txt, b.txt, c.txt each of size 1.25 Kb.

a.txt resides in blocks 0,1

b.txt resides in blocks 2,5

c.txt resides in block 3,4

The file system in this project consists of three parts as shown in the figure.

- Meta Header [Fixed size]
 - Number of file descriptors present
- Header [Fixed size]
 - File descriptors of each file/folder in the file system.
These file descriptors would be needed to populate n-array tree for the directory structure and hash table for faster searching for files.
 - There would be a separate free_block_list which contains the list of all the free blocks in the file system.
- File Blocks – Each identified with an index number or an offset.

Meta header (fixed size) (name of file system, number of used file descriptors)				
Header (max of N file descriptors + Free list) (fd_01, fd_02, ... fd_N)				
Block 0	Block 1	Block 2	Block n-1

Key Data Structures used

- N-ary tree
- Hash table
- Linked list
- Binary Search Tree
- Appropriate indexing structure for text-based search

Purpose of Data Structures

- N-ary tree would be used to represent the directory structure of the entire file system. Each node in the tree would be the file descriptor corresponding to the file/folder.
- Hash table would help in searching functionality for files on the basis of the name of the file. Each bucket in the hash would hold the files according to the starting alphabet of the file.
Example:

Key values	Buckets
a	aa.txt-->ab.ppt-->ac.doc
b	ba.txt-->bb.doc-->bc.ppt
c	ca.doc-->cb.txt-->cc.ppt
.	
.	
.	
z	za.txt-->zb.ppt-->zc.txt

- Linked list keeps a list of the free data block into which new files can be written.
- BST is used to perform search functionality on files on the basis of absolute path of the file.
- You need to explore and come up with suitable data structure for content-based search too

High Level VFS Operators

File System Operators

1. **createvfs (vfs_label,size)** – To create a Virtual File System. <vfs_label> is the name of VFS.
2. **mountvfs (vfs_label)** – To Mount the VFS path on current directory. This involves loading the file/directory descriptors into various data structures
3. **unmountvfs (vfs_label)** - To unmount the VFS. It involves saving index information and tree information back into the hard disk.

Directory Operators

4. **makedir (parent_path, dir_name)** – To create a new directory whose name should be <dir_name> in path specified by <parent_path> where '/' is considered as 'root' directory.
5. **deletedir (path)** – To remove a file or a directory as indicated by < name>
6. **movedir (source_dir_path,dest_dir_path)** – To move a sub-tree <source_dir_path> to <dest_dir_path> directory.
7. **listdir (dir_name,flag, outputfile)** – List all the contents according to <flag> of the current directory i.e. specified by <dir_name>. Flag “-R” means recursive listing of the entire subtree. The output of listdir is saved in outputfile in the hard-disk

File Operators

8. **addfile (dest_dir_name, file_name, data)** - Create a new file named <file_name> with <data> as content. in path specified by <dest_dir_name>. “data” corresponds to an external file residing in the hard disk.
9. **listfile (file_path, output_file)** - Extract the contents of a file specified by <file_path> and store in output file.
10. **updatefile (file_path, data)** – Update/Edit the contents of a file specified by <file_path> with <data>.
11. **removefile (file_path)** – Remove a file specified by <file_path>.
12. **movefile(source_file_path,dest_file_path)** – Rename <source_file_path> to <dest_file_path> file.
13. **copyfile (source_file/dir_path,dest_file/dir_path)** – To copy source directory/file to destination specified.
14. **exportfile (vfs_file_path, hard_disk_path)** – Export the file from the file system into the hard disk
15. **searchfile(name, outputfile)** – Retrieves the full path of all the files/folders having the given name as part of its file/folder name (substring). The output is stored in outputfile.

EVALUATION PROCEDURE

Your software will be evaluated using interaction scripts. Interaction scripts contain a series of VFS operators. A sample interaction script is shown below:

Command ID	Interaction command	Parameter Description	Appropriate output messages <i>Replace the contents shown in angular brackets with appropriate text from the program.</i>
1)	createvfs P1 P2	P1 – vfs_label P2 – size	createvfs_SUCCESS createvfs_FAILURE <errorcode>
2)	mountvfs P1	P1 – vfs_label	mountvfs_SUCCESS mountvfs_FAILURE <errorcode>
3)	unmountvfs P1	P1 – vfs_label	unmountvfs_SUCCESS unmountvfs_FAILURE <errorcode>
4)	makedir P1 P2	P1 – parent_path P2 – dir_name	makedir_SUCCESS makedir_FAILURE <errorcode>
5)	deletedir P1	P1 – directory path	deletedir_SUCCESS deletedir_FAILURE <errorcode>
6)	movedir P1 P2	P1 – source_dir_path P2 – dest_dir_path	movedir_SUCCESS movedir <errorcode>
7)	listdir P1 P2 P3	P1 – directory path P2 – flag (0 for non-recursive and 1 for recursive) P3 – text file path (from local hard disk)	listdir_SUCCESS listdir_FAILURE <errorcode>
8)	addfile P1 P2 P3	P1 – dest_dir_path P2 – file_name P3 – data file path	addfile_SUCCESS addfile_FAILURE <errorcode>
9)	listfile P1 P2	P1 – file path P2 – text file path (from local hard disk)	listfile_SUCCESS (in case text files only) listfile_FAILURE <errorcode>
10)	updatefile P1 P2	P1 – file path P2 – data file path (from local hard disk)	updatefile_SUCCESS updatefile_FAILURE <errorcode>

11)	removefile P1	P1 – file path	removefile_SUCCESS removefile_FAILURE <errorcode>
12)	movefile P1 P2	P1 – source_file_with_path P2 – destination path	movefile_SUCCESS movefile_FAILURE <errorcode>
13)	copyfile P1 P2	P1 – source_file_with_path P2 – destination path	copyfile_SUCCESS copyfile_FAILURE <errorcode>
14)	exportfile P1 P2	P1 – source_file_with_path P2 – destination file path (in local hard disk)	exportfile_SUCCESS exportfile_FAILURE <errorcode>
15)	searchfile P1 P2	P1 – name to be searched P2 – output file name	searchfile_SUCCESS <count of matches> searchfile_FAILURE <errorcode>

List of error codes will be given to you in the form of #defines in a file named vfs_errorcodes.h