

Project Report

Group Ravioli

Java and C# in depth, Spring 2014

Michal Bang

`mbang@student.ethz.ch`

Pascal Fischli

`fischlip@student.ethz.ch`

Vladimir Grozman

`grozmanv@student.ethz.ch`

April 7, 2014

1 Introduction

This document describes the design and implementation of the Personal Virtual File System of group ravioli. The project is part of the course Java and C# in depth at ETH Zurich. The following sections describe each project phase, listing the requirements that were implemented and the design decisions taken. The last section describes a use case of using the Personal File System.

2 VFS Core

VFS Core is the first step towards a Personal Virtual File System. It operates on virtual disks, of which each one is stored in a single file in the host file system. Its API not only offers functionalities to create, mount and delete virtual disks, but also to operate inside opened disks. This ranges from basic console operations, like navigating through directories, renaming and removing, to the virtual disk operations of importing and exporting, both on directories, files or both together.

The most important task in the process of creating the VFS Core is the design of the storage structure in the virtual disk files. The main aspect in this is efficiency. Not mainly in speed, but in usage of the available space. Two basic approaches are storing the files in one block, which of course

could result in lots of space lost due to fragmentation, or splitting files into parts of predefined size that are always linking to the next part, resulting in a very space efficient but most likely slower system.

2.1 Requirements

In this part we indicate and describe what requirements we have implemented. We also list the main software elements involved in the respective implementation.

1. The virtual disk is stored in a single file in the host file system. The creation happens in the creators of JCDVFS, which are creating a `FileStream` for the new file that is then written to in the creators of JCDFAT.
2. The creation of virtual disk files happens in method `Create(string hfsPath, ulong size)` of class JCDVFS. The parameters specify the location and the initial size.
3. Several virtual file systems on the host are allowed. Creation and opening, which is done with method `Open(string hfsPath)` of class JCDVFS, can happen an unlimited amount of times and with the file at any place on the host file system.
4. Virtual disks can be disposed from the host file system, which can be done through method `Delete(string hfsPath)` of class JCDVFS. Only files containing a virtual disk that can be opened are disposable through this.
5. On a mounted VFS files and directories can be created, deleted and renamed with the methods `CreateFile(ulong size, string path, bool isFolder)`, `DeleteFile(string path, bool recursive)` and `RenameFile(string vfsPath, string newName)` in class JCDFAT.
6. For basic navigation functionalities going to a specific location can be done with method `SetCurrentDirectory(string path)` in class JCDFAT. To support that, listing of files and directories can be done with the method `ListDirectory(string vfsPath)` in class JCDFAT, which is called by the method with the same signature in class JCDVFS.
7. Moving of files and directories can be done with method `MoveFile(string vfsPath, string newVfsPath)` and copying with method `CopyFile(string`

`vfsPath, string newVfsPath)` in JCDFAT. Both preserve the internal file/directory hierarchy.

8. Files and directories can be imported from the host into the virtual file system. This is done through method `ImportFile(string hfsPath, string vfsPath)` in JCDFAT which is then calling `ImportFolder(string hfsFolderPath, string vfsPath)` or `ImportFile(Stream file, string path, string fileName)` in JCDFAT.
9. Exporting of files and directories from the virtual to the host file system is done in methods `ExportFile(Stream outputFile, JCDFFile file)` and `ExportFolderRecursive(JCDFFolder folder, string hfsPath)` in JCDFAT.
10. How much space is free in the virtual disk can be queried and is calculated in method `GetFreeSpace()` in JCDFAT, which is multiplying the number of free blocks times the block size. The occupied space can be queried too and is calculated in method `OccupiedSpace()` in JCDFAT which is subtracting the free space from the virtual disk size.

2.2 Design

2.3 Unit Tests

After having defined the API and prior to implementing its functionalities we have created a set of unit tests. For each method in the interface we have identified the "normal", and also the possible "wrong" use cases and the expected reaction of the system to them. Each such use case resulted in an own test, which can be run independently of the others. The test framework that we have used is MSTest.

Since we are running the tests on the same files that are created and then deleted for each test run, our tests cannot be run or analysed for code coverage in parallel.

5 Quick Start Guide

To facilitate debugging we have created a console client. The available commands are described in this section.

1. Can be called all the time:
 - `help`: Shows the help text with the available commands.

- exit/quit: Exit the client.
2. Can be called only when NOT mounted:
 - create path size: Create a new VFS file at the given path and with the given size.
 - delete path: Delete the VFS file at the given location.
 - open path: Open/mount the VFS file at the given location.
 3. Can be called only when a VFS is mounted:
 - close: Close the mounted VFS.
 - ls [path]: List the files/directories in the current or in the given directory.
 - cd path: Change to the given directory.
 - rm path [-r]: Remove the given file/directory (recursively with -r).
 - mk path [-p]: Make a new file (and parents with -p).
 - mkdir path [-p]: Make a new directory (and parents with -p).
 - cp source target: Copy the source to the target, both on the VFS.
 - mv -hv/-vh/-vv source target: Move the source to the target. In the first parameter the first letter indicates the source and the second the target file system. h stands for host and v for the virtual. Therefore -hv means import, -vh export and -vv inside the VFS.
 - rn path newName: Rename the file/directory to the given new name.
 - size: Show the size of the hole VFS file.
 - free: Show the free space.
 - occupied: Show the occupied space.

References