Group Project: In-Memory Virtual File System

Department of Computing

September 14, 2020

Deadlines

1. Archive the source code, the tests, the presentation slides, and the final report in PDF into a ZIP file, and submit it on Blackboard before or on November 29, 2020.
2. Submit the design review report on Blackboard before or on November 30, 2020.

1 Overview

The goal of this project is to develop an in-memory Virtual File System (VFS), named the Comp VFS (CVFS), in Java. A VFS is usually built on top of a host file system to enable uniform access to files located in different host file systems, while the CVFS simulates a file system in memory. You can find more information about VFSs on Wikipedia[1].

2 Requirements

The CVFS should satisfy the following requirements: The system operates on one virtual disk at a time; Each virtual disk has a maximum size[2] and may contain as many files, i.e., documents and directories, as allowed by that size; For each document, the virtual disk maintains its `name`, `type`, and `content`, all of type `String`; For each directory, the virtual disk maintains its `name` of type `String` and the list of files *directly* contained in it; Only digits and English letters are allowed in file names, and each file name may have at most 10 characters; Only documents of types `txt`, `java`, `html`, and `css` are allowed in the system; The size of a document is calculated as `40+content.length*2`, and the size of a directory is calculated as 40 plus the total size of its contained files[3].

The CVFS should also provide a command line interface (CLI) tool to facilitate the use of virtual disks, and some of the requirements for the CLI tool are as the following. Here we use symbol '`$`' to denote the working directory and symbol '`:`' to separate the file names in a path. For example, path "`$:xyz`" refers to a file named "`xyz`" inside the working directory. Depending on whether the file actually exists in the working directory or not, the path may be valid or invalid.

[REQ1] (2 points) The tool should support the creation of a new disk with a specified maximum size.
Command: `newDisk` *diskSize*
Effect: Creates a new virtual disk with the specified maximum size. The previous working disk, if any, is closed; The newly created disk is set to be the working disk of the system, and the working directory is set to be the root directory of the disk.

[REQ2] (2 points) The tool should support the creation of a new document in the working directory.
Command: `newDoc` *docName docType docContent*

[1]`https://en.wikipedia.org/wiki/Virtual_file_system`
[2]All sizes in the requirements are in bytes.
[3]Such simplification is to make the calculation independent of specific implementations.

Effect: Creates a new document in the working directory with the specified name, type, and content.

[REQ3] (2 points) The tool should support the creation of a new directory in the working directory.
Command: `newDir` *dirName*
Effect: Creates a new directory in the working directory with the specified name.

[REQ4] (2 points) The tool should support the deletion of an existing file in the working directory.
Command: `delete` *fileName*
Effect: Delete an existing file with the specified name from the working directory.

[REQ5] (2 points) The tool should support the rename of an existing file in the working directory.
Command: `rename` *oldFileName newFileName*
Effect: Rename an existing file in the working directory from *oldFileName* to *newFileName*.

[REQ6] (2 points) The tool should support the change of working directory.
Command: `changeDir` *dirName*
Effect: If there is a directory with the specified name in the working directory, use that directory as the new working directory; If *dirName* is "`..`", i.e., two dots, and the working directory is not the root directory of the working disk, use the parent directory of the working directory as the new working directory.

[REQ7] (4 points) The tool should support listing all files direclty contained in the working directory.
Command: `list`
Effect: List all the files directly contained in the working directory. For each document, list the name, type, and size; For each directory, list the name and size. Report the total number and size of files listed.

[REQ8] (4 points) The tool should support *recursively* listing all files in the working directory.
Command: `rList`
Effect: List all the files contained in the working directory. For each document, list the name, type, and size; For each directory, list the name and size. Use indentation to indicate the level of each file. Report the total number and size of files listed.

[REQ9] (3 points) The tool should support the construction of simple criteria.
Command: `newSimpleCri` *criName attrName op val*
Effect: Construct a simple criterion that can be referenced by *criName*. A *criName* contains exactly two English letters, and *attrName* is either `name`, `type`, or `size`. If *attrName* is `name`, *op* must be `contains` and *val* must be a string in double quote; If *attrName* is `type`, *op* must be `equals` and *val* must be a string in double quote; If *attrName* is `size`, *op* can be `>`, `<`, `>=`, `<=`, `==`, or `!=`, and *val* must be an integer.

[REQ10] (2 points) The tool should support a simple criterion to check whether a file is a document.
Criterion name: `IsDocument`
Effect: Evaluates to `true` if and only if on a document.

[REQ11] (3 points) The tool should support the construction of composite criteria.
Command: `newNegation` *criName1 criName2*
Command: `newBinaryCri` *criName1 criName3 logicOp criName4*
Effect: Construct a composite criterion that can be referenced by *criName1*. The new criterion constructed using command `newNegation` is the negation of an existing criterion named *criName2*; The new criterion constructed using command

`newBinaryCri` is *criName3 op criName4*, where *criName3* and *criName4* are two existing criteria, while *logicOp* is either `&&` or `||`.

[REQ12] (4 points) The tool should support the printing of all defined criteria.
Command: `printAllCriteria`
Effect: Print out all the criteria defined. All criteria should be resolved to the form containing only *attrName*, *op*, *val*, *logicOp*, or `IsDocument`.

[REQ13] (4 points) The tool should support searching for files in the working directory based on an existing criterion.
Command: `search` *criName*
Effect: List all the files directly contained in the working directory that satisfy criterion *criName*. Report the total number and size of files listed.

[REQ14] (4 points) The tool should support *recursively* searching for files in the working directory based on an existing criterion.
Command: `rSearch` *criName*
Effect: List all the files contained in the working directory that satisfy criterion *criName*. Report the total number and size of files listed.

Note that some important operations for the CLI tool are omitted here. You do *not* need to take those operations into account when designing and implementing the system. Also note that the requirements above do not stipulate what should happen when a command is in bad format or the required action cannot be done, e.g., because a file name is invalid or the working disk is running out of space. You need to use your best judgment to gracefully handle those situations. Poor design in this aspect will lead to point deductions.

The CLI tool may be extended with the following *bonus* features:

[BON1] (8 points) Support for `store` and `load` commands that store/load a virtual disk to/from the local file system.

[BON2] (8 points) Support for `undo` and `redo` commands.

3 Group Forming

This is a group project. Each group may have 3 to 4 members. Please team up on Blackboard/Groups before or on September 25, 2020. Afterwards, we will randomly team up the ones with no group, and whoever requests for changing the group needs to

1. seek agreements from all the members in both his/her current group and the group that he/she wants to join, and
2. inform the instructor via email of the desired change, with agreements from the other members attached. The email should be cc-ed to all the relevant members. Upon receiving this email, we will make the corresponding change.

4 Code Inspection

The inspection tool of IntelliJ IDEA[4] checks your program to identify potential problems in the source code. We have prepared a set of rules to be used in grading (`COMP2021_PROJECT.xml` in the project material package). Import the rules into your IntelliJ IDEA IDE and check your implementation against the rules. Unit tests do not need to be checked.

The inspection tool is able to check for many potential problems, but only a few of them are considered errors by the provided rule set. 2 points will be deducted for each *error* in your code reported by the tool.

[4]https://www.jetbrains.com/help/idea/code-inspection.html

5 Statement Coverage of Unit Tests

Statement coverage[5] is a measure used in software testing to report the percentage of statements that have been tested: The higher the coverage, the more thoroughly we have tested a program.

You should follow the Model-View-Controller (MVC) pattern[6] in design the CVFS. Put all Java classes for the model into package `hk.edu.polyu.comp.comp2021.cvfs.model` (see the structure in the sample project) and write tests for the model classes.

During grading, we will use the coverage tool of IntelliJ IDEA[7] to collect statement coverage information on your tests for the `model` subpackage. Your will get 10 base points for a statement coverage between 90% and 100%, 9 base points for a coverage between 80% and 89%, and so on. You will get 0 base points for a statement coverage below 30%. The final points you get for statement coverage of unit tests will be calculated as your base points multiplied by the percentage of your requirement coverage. For example, if you only implement 60% of the requirements and you achieve 95% statement coverage in testing, then you will get 6 = 10 * 60% points for this part.

6 Design Review

The 3-hour lecture time in Week 13 will be devoted to the peer review of your designs. The review will be conducted in clusters of three to four groups; Each group needs to review the designs of the other groups in its cluster and fill out the review reports. In a review report, a group needs to point out which design choices from another group are good, which are questionable, and which should have been avoided, and then explain the reasons.

Details about the organization of the design review and the format of the review report will be announced before the review session.

7 Project Grading

You can earn at most 100 points in total in the project from the following components:

- Requirements coverage (in total 40 points, as listed in Section 2)
- Code quality (10 points for good object-oriented design and 10 for good code style, as reported by the code inspection tool)
- Statement coverage of unit tests (10 points)
- Presentation (7 points)
- Design review report (8 points)
- Final report (15 points)
- Bonus points (16 points)

  Note: Bonus points can be used to reach, but not exceed, 100 points.

In the project, each group member is expected to actively participate and make fair contributions. In general, members of a group will receive the same grade for what their group produces at the end. *Individual grading*, however, could be arranged if any member thinks "one grade for all" is unfair and files a request to the instructor in writing (e.g., via email). In individual grading, the grade received by each group member will be based on his/her own contributions to the project, and each member will need to provide evidence for his/her contributions to facilitate the grading.

[5] http://en.wikipedia.org/wiki/Code_coverage
[6] https://en.wikipedia.org/wiki/Model-view-controller
[7] https://www.jetbrains.com/help/idea/code-coverage.html

8 General Notes

- Java SE Development Kit Version 8[8] and IntelliJ IDEA (Community Edition) Version 2020.1[9] will be used in grading your project. Make sure you use the same versions of tools for your development.
- Your code should not rely on any library that is not part of the standard Java SE 8 API.
- The project material package also include three other files:
  - `SampleProject.zip`: Provides an example for the organization of the project. Feel free to build the CVFS based on the sample project.
  - `IntelliJ IDEA Tutorial.pdf`: Briefly explains how certain tasks necessary for the project can be accomplished in IntelliJ IDEA.
  - `COMP2021_PROJECT.xml`: Contains the rules to be used in code inspection.

If you use other IDEs for your development, make sure all your classes and tests are put into an IntelliJ IDEA project that is ready to be tested and executed. 50 points will be deducted from projects not satisfying this requirement or with compilation errors!

[8] https://www.oracle.com/hk/java/technologies/javase/javase-jdk8-downloads.html
[9] https://www.jetbrains.com/idea/download/other.html

9 Project Report Template

Project Report
Group XYZ
COMP2021 Object-Oriented Programming (Fall 2020)
Member1
Member2
Member3

1 Introduction

This document describes the design and implementation of the Comp Virtual File System (CVFS) by group XYZ. The project is part of the course COMP2021 Object-Oriented Programming at PolyU.

2 The Comp Virtual File System (CVFS)

In this section, we describe first the overall design of the CVFS and then the implementation details of the requirements.

2.1 Design

Use class diagram(s) to give an overview of the system design and describe in general terms how different components fit together. Feel free to elaborate on design patterns used (except for the MVC pattern) and/or anything else that might help others understand your design.

2.2 Requirements

For each (compulsory and bonus) requirement, describe 1) whether it is implemented and, when yes, 2) how you implemented the requirement as well as 3) how you handled various error conditions.

[REQ1] 1) The requirement is implemented.
2) Implementation details.
3) Error conditions and how they are handled.

[REQ2] 1) The requirement is not implemented.

[REQ3] ...

3 User Manual

In this section, we explain how the CVFS works from a user's perspective.

Describe here all the commands that the system supports. For each command, use screenshots to show the results under different inputs.