Group Project: In-Memory Virtual File System

Department of Computing

September 12, 2024

Deadlines

1. Form your groups of 3 to 4 students before 14:00, 26 September 2024. Afterward, students with no group will be randomly grouped. Requests for group change will be approved only if written agreements from all members of the involved groups are provided before 14:00, 30 September 2024.
2. Compress the source code (including the tests), the tests, the user manual, the presentation slides and recording, and the project report into a single ZIP archive and submit it on Blackboard before 14:00, 21 November 2024.
3. Submit the design review reports in PDF or JPG format on Blackboard before 14:00, 28 November 2024.
4. Late submissions will lead to 25% deduction of points per day.

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 may find more information about VFSs on Wikipedia[1]. Or, you may search the related information from diverse sources.

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, and all the documents and directories have unique names; only documents of types `txt`, `java`, `html`, and `css` are allowed in the system; document types are not part of document names; 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 the symbol '$' to denote the working directory and the symbol ':' to separate the file names in a path.~~

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

~~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*
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] (3 points) The tool should support listing all files directly 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] (3 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 the double quote; If *attrName* is `type`, *op* must be `equals` and *val* must be a string in the

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 the command `newNegation` is the negation of an existing criterion named *criName2*; The new criterion constructed using the command `newBinaryCri` is *criName3 op criName4*, where *criName3* and *criName4* are two existing criteria, while *logicOp* is either `&&` or `||`.

[REQ12] (3 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] (3 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] (3 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 the criterion *criName*. Report the total number and size of files listed.

[REQ15] (2 points) The tool should support saving the working virtual disk (together with the files in it) into a file on the local file system.
Command: `save` *path*
Effect: Save the working virtual disk into the file at *path*.

[REQ16] (2 points) The tool should support loading a virtual disk (together with the files in it) from a file on the local file system.
Command: `load` *path*
Effect: Load a virtual disk from the file at *path* and make it the working virtual disk.

[REQ17] (1 point) The user should be able to terminate the current execution of the system.
Command: `quit`
Effect: Terminate the execution of the system.

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 handle those situations gracefully. Poor design in this aspect will lead to point deductions.

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

[BON1] (6 points) Support for saving and loading the defined search criteria when a virtual disk is saved to/loaded from the local file system.

[BON2] (6 points) Support for the `undo` and `redo` of commands `newDoc`, `newDir`, `delete`, `rename`, `changeDir`, `newSimpleCri`, and `newNegation`.

3 Group Forming

This is a group project. Each group may have 3 to 4 members from the same class session, i.e., *no group across different class sessions is allowed*. Form your group before 14:00, 26 September 2024. Afterward, students with no group will be randomly grouped, and requests for group change are only approved if written agreements from all members of the involved groups are provided before 14:00, 30 September 2024.

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.

5 Line Coverage of Unit Tests

The line coverage[5] achieved by a suite of unit tests is the percentage of source code lines that the tests exercise, and it is an important measure of the test suite's quality. The higher the coverage, the more thoroughly the suite of tests exercised a program.

You should follow the Model-View-Controller (MVC) pattern[6] in designing 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 get the line coverage of your tests on package `hk.edu.polyu.comp.comp2021.tms.model`. You will get 10 base points for line coverage between 90% and 100%, 9 base points for coverage between 80% and 89%, and so on. You will get 0 base points for line coverage below 30%. The final points you get for line coverage of unit tests will be calculated as your base points multiplied by the percentage of your requirements coverage. For example, if you only implement 60% of the requirements and you achieve 95% line coverage in testing, 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 design. The review will be conducted in clusters of X groups (X is to be decided later); Every group needs to review the design of each other group in its cluster and fill out a review report. That is, every group will need to fill out and submit X-1 review reports. In each report, the reviewer group needs to point out which design choices from the reviewee group are good, which are questionable, which should have been avoided, and explain the reasons.

Details about the design review's organization and the report's format will be announced before the review.

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

7 Project Report

Each group must submit a project report that summarizes the group's work in the project, and you may refer to Section A for the required structure and contents for the report.

Individual contributions: In particular, your group needs to reach a consensus on the percentage of contributions each member has made to the project, and the percentages need to be clearly stated right below the report title. For example, if your group has 3 members and they have made equal contributions, you may write that each member has completed 33.3% of the work. Refer to Section 10 for how this percentage influences your individual score for the project.

If your group has problems in reaching such a consensus, you should submit all the other required materials before the deadline and report your case to the instructor before 14:00, 22 November 2024.

Use of GenAI tools: Using GenAI tools in the project is permitted, but you should give proper acknowledgment to all the content from those tools that you included in your deliverables.

If you use GenAI tools in preparing any of the deliverables, you need to clearly state the following in the subsection titled "My Contributions and Role Distinction" of your project~~individual~~ report: 1) Which deliverable(s) and which part(s) of the deliverable(s) were prepared with the help of GenAI tools; 2) How you used the GenAI tools; 3) What you did to verify/revise/improve the results produced by the tools. Without such explanations in your report, we will consider all the deliverables of your original work.

Failure to reference externally sourced, non-original work can be considered plagiarism. You may refer to `https://libguides.lb.polyu.edu.hk/academic-integrity/GenAI_and_Plagiarism` for more guidance on GenAI and plagiarism.

8 Presentation

The presentation should not exceed 6 minutes, and each group member must present at least 1 minute.

Your presentation should at least include the following parts: 1) the overall architecture of your program, 2) a design choice for the program that concerns inheritance and polymorphism, and 3) a short discussion on how object-oriented design/programming helped you improve the reusability/scalability of the code. You may also include screen snapshots or short video clips of the CVFS running in your presentation, but do not let them dominate or excessively consume the time allocated.

Your presentation video should be in MP4 format. To facilitate validating the presenters, all presenters are required to show their student ID cards and faces at the beginning of the presentation, and the current presenter's face must always be shown in the video.[8] Each group member who does not show his/her face or student ID card in the presentation will have half of his/her marks deducted.

9 Submission

Each group is required to submit a single ZIP file containing their source code (including the tests), the user manual, the presentation slides and video, and the project~~individual~~ report~~s~~. You should organize the files and directories in a consistent structure to ensure clarity and ease of access. ~~It is important to note that individual reports of group members should be included in the "Reports" directory, with student IDs incorporated into the file names as identifiers.~~ Here are the guidelines for preparing the ZIP file:

1. Create a main directory with your project's name or a suitable identifier.

[8] All presentation videos will be automatically destroyed in the system after one year to protect your privacy.

2. Within the main directory, create subdirectories for each component of the submission, such as SourceCode, UserManual, Reports, and Presentation.
3. Place the relevant files in the required format in their corresponding directories. For example:
   - The project report should be placed inside the main directory.
   - The user manual document should be placed inside the main directory.
   - The IntelliJ IDEA project folder should be placed inside the SourceCode directory.
   - The presentation slides and video should be placed inside the Presentation directory.
4. Select all the directories and files within the main directory.
5. Compress the selected items into a ZIP file.

Example Directory Structure:

```
GroupID
├── project_report.pdf
├── user_manual.pdf
├── SourceCode
│   └── The IntelliJ IDEA project folder
└── Presentation
    ├── slides.pdf
    └── video.mp4
```

10 Grading

The overall project work of your group can earn at most 100 points in total 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)
- Line coverage of unit tests (10 points)
- Presentation (7 points)
- Design review report (8 points)
- Project~~Individual~~ report and user manual (15 points)
- Bonus points (12 points)

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

Your individual score for the project will be calculated based on the score for the overall project work of your group and your percentage of contributions to the project. Suppose the overall project work of a group is worth x points, the group has y members ($2 < y < 5$), and one member's contribution is $z\%$. The member's individual score for the project is calculated as $\min(x * y * z\%, x)$. We encourage equal contribution.

11 General Notes

- Java SE Development Kit Version 21[9] and IntelliJ IDEA **Community Edition** (Version 2024.2[10]) will be used in grading your project. Make sure you use the same versions of tools in your development. Note that you need to configure "File | Project Structure... | Project Settings | Project" in the IntelliJ IDEA IDE as the following:
  - SDK: 21
  - Language level: 8 - Lambdas, type annotations etc.
- Your code should not rely on any library beyond the standard Java SE Development Kit 21 API.
- The project material package also includes three other files:
  - `SampleProject.zip`: Provides an example for the structure of the project directory. Feel free to build the interpreter based on the sample project.
  - `IntelliJ IDEA Tutorial.pdf`: Briefly explains how certain tasks relevant to the project can be accomplished in IntelliJ IDEA.
  - `COMP2021_PROJECT.xml`: Contains the rules for code inspection.

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

[9] https://www.oracle.com/java/technologies/javase/jdk21-archive-downloads.html
[10] https://www.jetbrains.com/idea/download/other.html

Appendixes

A Project Report Template

Project Report
COMP2021 Object-Oriented Programming (Fall 2024)
Group XYZ

Members and contribution percentages:
Member1: a%
Member2: b%
Member3: c%
Member4: d%

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 explain how different components fit together. Feel free to elaborate on the used design patterns and/or anything else that might help others understand your design.

2.2 Implementation of 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 error handling.

[REQ2] 1) The requirement is not implemented.

[REQ3] ...

3 Reflection on My Learning-to-Learn Experience

Some requirements in this project concern aspects of Java and/or object-oriented programming that are not fully covered in lectures, and you need to plan, conduct, evaluate, and adjust your self-learning activities to master the related knowledge and accomplish the corresponding tasks.

Use this section to 1) reflect on your learning-to-learn experience in the project and 2) describe your plan to improve your self-learning approaches.

B User Manual

To prepare a user manual, it's important to note that there is no one-size-fits-all template. User manuals can differ significantly based on the system being documented and the specific needs of the users. However, the following guidelines can help you create an effective user manual:

- Introduction: Begin by providing an overview of the system and its purpose. Explain how the system works from a user's perspective.
- Commands Description: Describe all the commands that the system supports. Provide detailed explanations of each command, including their functionalities and how they can be used.
- Screenshots: For each command, consider including screenshots to visually demonstrate the results under different inputs. Capture screenshots that showcase the system's response or output when specific commands are executed. Include captions or annotations to provide additional context or explanations for each screenshot.
- Step-by-Step Instructions: Provide clear and concise step-by-step instructions for users to follow when using each command. Break down the process into logical steps, ensuring that users can easily understand and replicate the actions.
- Troubleshooting: Include a troubleshooting section to address common issues or errors that users may encounter while using the system. Provide solutions or workarounds for each problem, ensuring that users can resolve issues on their own.
- Additional Resources: If applicable, include additional resources such as references, FAQs, or contact information for technical support. Provide users with avenues to seek assistance or further information if needed.
- Formatting and Organization: Ensure that the user manual is well-formatted and organized. Use headings, subheadings, and bullet points to improve readability. Consider using a consistent and intuitive structure throughout the manual.

Remember, the purpose of a user manual is to provide clear instructions and support to users. Tailor the manual to meet the specific needs of your system and its users.