Assignment 2: Virtual Filesystem

Data Structures (CS-UH 1050) — Spring 2023

1 Code of Conduct

All assignments are graded, meaning we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we will briefly state what is and isn't allowed when working on an assignment.
Any documents and program code that you submit must be fully written by yourself. You can discuss your work with fellow students, as long as **these discussions are restricted to general solution techniques, without sharing the overall or specific algorithms.** Put differently, these discussions should not be about concrete code you are writing, nor about specific set of steps or results you wish to submit. When discussing an assignment with others, this should never lead to you possessing the complete or partial solution of others, regardless of whether the solution is in paper or digital form, and independent of who made the solution, meaning you are also not allowed to possess solutions by someone from a different year or course, by someone from another university, or code from the Internet, etc. This also implies that **there is never a valid reason to share your code with fellow students, and that there is no valid reason to publish your code online in any form as long as you are a student at NYUAD**. Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g., if the solution matches with high similarity score that of others), **the suspected violations will be reported to the academic administration according to the policies of NYU Abu Dhabi** (see https://students.nyuad.nyu.edu/campus-life/community-standards/policies/academic-integrity/) under the integrity review process.

2 Introduction

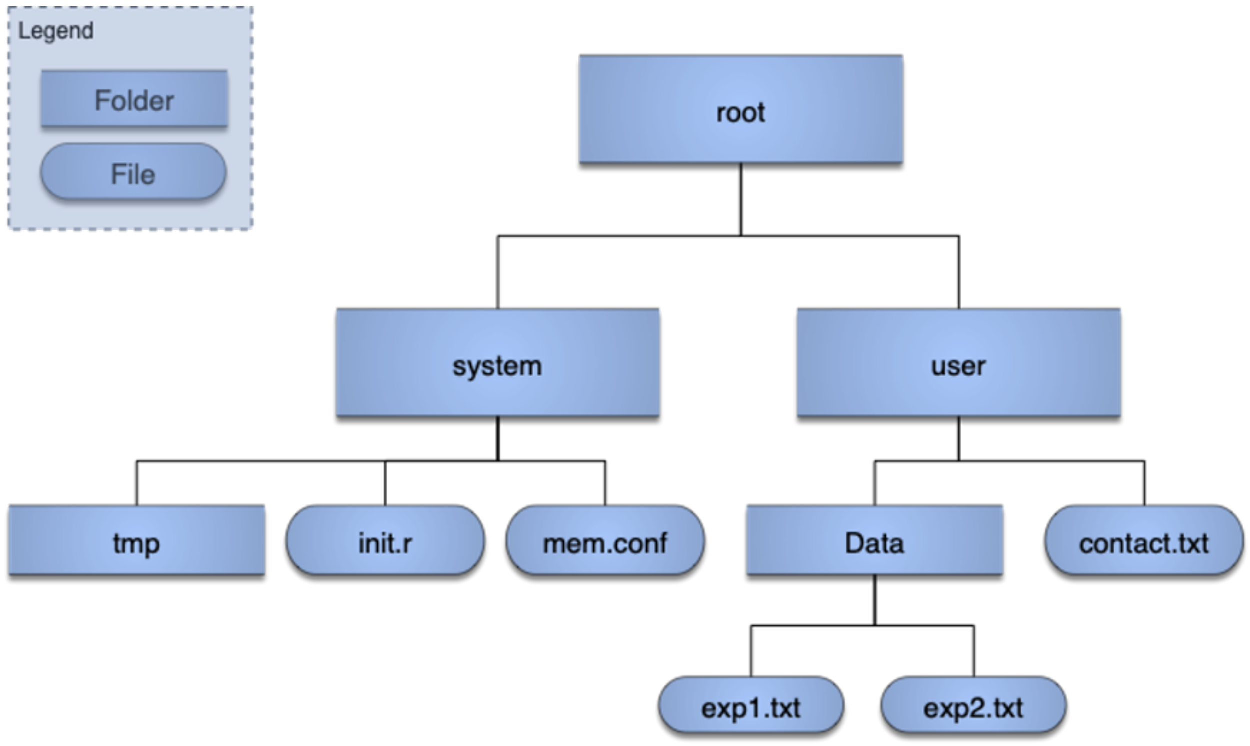
In this assignment, you will develop a virtual filesystem (VFS). A user can create/delete/move folders and files, among other operations described below. **The filesystem should be represented and organized as a Tree**. Each **inode**[1] contains metadata about the node (e.g., file or folder, size, creation date). A folder can have zero or many files and folders, while a file is considered as a leaf node only. Deleting a file puts the inode's element in a limited size queue (the Bin/Trash) which can be emptied either manually or automatically. The user interface should allow a user to navigate and perform the described tasks below. At the start and end of the session, the file system is loaded, and dumped to the disk, respectively.
You are supposed to create a program utilizing the object-oriented programming (OOP) principles and appropriate data structures discussed in class. **You should implement all the data structures manually, STL based containers are not allowed to be used here.**

[1] This is a reference to the actual unix style filesystem: https://en.wikipedia.org/wiki/Inode

3 Implementation

Overview:

Here is a visual example of the virtual filesystem at a given state.



Storing the above state on disk will generate the following file:

vfs.dat

```
/,0,12-02-23
/system,10,12-02-23
/system/tmp,10,12-02-23
/system/init.r,150,12-02-23
/system/mem.conf,512,12-02-23
/user,10,12-02-23
/user/Data,10,01-03-23
/user/Data/exp1.txt,1886,01-03-23
/user/Data/exp2.txt,22232,01-03-23
/user/contact.txt,45,15-02-23
```

Constraints:

- The root inode has an empty element, and cannot be deleted. Printing the path of the root returns “/”.
- A file is an external inode, and cannot have subfolders.
- Filenames or Folder names under a given inode must be unique.
- Each inode has a name, size and a date (extracted from system time).
- File names should be alphanumeric only (i.e., comprises the letters A to Z, a to z, and the digits 0 to 9) without whitespaces or special characters, except the period “.” that can be used prior to file extensions.
- The date of a file is its creation date.
- The default size of a **folder inode** is **10 bytes**, except for the **root** whose size is **0 byte**.
- The **total size of a folde**r is the sum of all of its descendants’ sizes (including its own size).
- The size of a file is user specified value at creation time (see the *touch* command).
- The path of an inode is the series of names of its ancestors including its own name. Each name is preceded by a slash symbol “/”.
- **DO NOT** store the full path in an inode.

Commands to implement:

When your program starts, the user is greeted and presented with a prompt to enter one of the commands below. By default, the system is initiated at the root of the virtual filesystem. The user is able to interact with the system using the commands until an exit command is issued.

```
=======================================================================================================================
Welcome to the Virtual Filesystem!

List of available Commands:
pwd                        : Print full path to the current working directory
realpath <filename>        : Print the absolute/full path of a given filename of a file within the current directory
ls                         : List files and folders inside the current folder
mkdir <foldername>         : Make a new directory/folder
touch  <filename> <size>               : Create a new file
find <filename|foldername>             : Return the path of the file (or the folder) if it exists
cd <filename>                          : Change directory
mv <filename> <foldername>             : Move a file
rm <filename|foldername>               : Remove a file or a folder
size <filename|foldername>             : Print size of an file/folder
find <filename|foldername>             : Return the path of the file (or the folder) if it exists
emptybin                               : Empty the trash
showbin                    : Print the oldest file/folder of the bin
recover                    : Recover the oldest file/folder form bin if path still exists
help                       : Display the list of available commands
exit                       : Exit the Program
=======================================================================================================================
```

When the program starts, it loads a pre-existing filesystem structure from vfs.dat (see visual example above). If the file does not exist, assume starting from just a root inode.

As a user navigates through the file system, you must keep track of the **current** **inode** folder location.

Note: in ***blue*** *are user specified names or parameters*

1. **help**
   - Prints the following menu of commands
2. **pwd**
   - Prints the path of current inode
   - Hint: Use a Stack based implementation
3. **realpath filename**
   - Prints the absolute/full path of a given filename of a file within the current directory
4. **ls**
   - Prints the children of the current inode, if folder (error otherwise)
   - Each line shows: filetype (dir/file), filename, size, date
   - **ls sort**: order by descending file size, use bubble sort

5. **mkdir foldername**
   - Creates a folder under the current folder
6. **touch filename size**
   - Creates a file under the current inode location with the specified filename, size, and current datetime
7. **cd**
   - **cd foldername**: change current inode to the specified folder
   - **cd filename**: return an error
   - **cd ..** changes current inode to its parent folder
   - **cd -** changes current inode to the previous working directory
   - **cd** changes current inode to root
   - **cd /my/path/name** changes the current inode to the specified path if it exists
8. (**find foldername**) or (**find filename**)
   - Returns the path of the file (or the folder) if it exists
   - You should print all matching paths. The same name may exist in different locations
   - Starts the search from the root folder
9. **mv filename foldername**
   - Moves a file located under the current inode location, to the specified folder path
   - The specified file and folder have to be one of the current inode's children (an error is returned otherwise)
10. (**rm foldername**) or (**rm filename**)
    - Removes the specified folder (along with its descendants) or file and puts it in a Queue of MAXBIN=20
    - The specified file or folder has to be one of the current inode's children (an error is returned otherwise)
    - Hint: Use a Queue based implementation

11. **[Bonus Feature]** Implement **mv** and **rm** on *arbitrary* inode locations
    - The inode and destination folder are specified using a path
    - Examples:
        - **mv /user/contact.txt /system/tmp**
        - **mv /user/tmp /user/**
        - **rm /user/contact.txt**
        - **rm /user/tmp**
12. **size foldername** or **filename**
    - Returns the total size of the folder, including all its subfiles, or the size of the file
    - The specified file or folder doesn't have to be one of the current inode's children
13. **emptybin**
    - Empties the bin
14. **showbin**
    - Shows only the oldest inode of the bin, including its path
15. **recover**
    - Reinstates the oldest inode back from the bin to its original position in the tree along with its descendants at the time of deletion (if the path doesn't exist anymore, an error is returned)
16. **Exit**
    - The program stops and the filesystem is saved in the format of the vfs.dat example

Example:

The aforementioned example filesystem **vfs.dat** was created from scratch using the following commands:

```
> mkdir system
> mkdir user
> cd system
> mkdir tmp
> touch init.r 150
> touch mem.conf 512
> cd ..
> cd user
> mkdir Data
> touch contact.txt 45
> cd Data
> touch exp1.txt 1886
> touch exp2.txt 22232
```

Additional operations are demonstrated below:

```
> find exp1.txt
/user/Data/exp1.txt
> size /
123123 bytes
> size /user/Data
24865 bytes
> cd /user/Data
> pwd
/user/Data
> rm exp1.txt
> showbin
/user/Data/exp1.txt (1886 bytes, 01-03-21)
> emptybin
> showbin
The bin is empty
> cd
> pwd
/
> ls
dir  system 12-02-23 10bytes
dir  user 12-02-23 10bytes
> cd -
> pwd
/user/Data
> ls
dir Data 01-03-23 10bytes
file contact.txt 15-02-23 45bytes
> ls sort
file contact.txt 15-02-23 45bytes
dir Data/ 01-03-23 10bytes
> size /user/Data
22232 bytes
> mv contact.txt Data
> ls
dir Data 01-03-23 10bytes
> size /user/Data
22277 bytes
> cd Data
> ls
file contact.txt 15-02-23 45bytes
file exp2.txt 01-03-23 22232bytes
```

3 Grading

Description	**Score (/20)**
- Loading, reading, dumping the system files properly - Defining ALL the needed classes correctly, with their sets of attributes and methods (including the constructors and destructors), and creating objects from each class properly - Proper initiation and termination of the system - Using the appropriate (most efficient) implementations for any data structure, e.g., vectors/lists/tree/queue/stack etc.	3
Main methods to design and implement: pwd (1 point), realpath (1 point), ls (1.5 point), mkdir (1 point), touch (1 point), cd (1.5 point), find (1 point), mv (1 point), rm (1 point), size (1.5 point), emptybin (1 point), showbin (1 point), recover (1.5 point).	15
Proper error handling of missing and invalid input, etc.	1
Properly commenting the code (i.e., code documentation)	1
[Bonus Feature] Implement **mv** and **rm** on *arbitrary* inode locations properly	1

You should solve and **work individually** on this assignment. The deadline of this assignment is **in 12 days of its release** on NYU Brightspace.

You should compress your **C++ source files** (*.cpp, *.h, makefile) in to **a single zip file** before uploading it directly to NYU Brightspace.

NO SUBMISSIONS OR RESUBMISSIONS VIA EMAIL WILL BE ACCEPTED. Note that your program should be implemented in C++ and **must be runnable on the Linux, Unix or macOS operating system**. Late submissions will be accepted **only up to 2 days late**, afterwards you will receive zero points. For late submissions, 5% will be deducted from the homework grade per late day.

Make sure to **FOLLOW THE ASSIGNMENT SUBMISSION AND EXTENSION PROTOCOLS** that are shared as part of the syllabus, the course logistics and Lecture#1 on NYU Brightspace, in addition to **THE ACADEMIC INTEGRITY POLICY** shared via the code-of-conduct, syllabus, the course logistics, Lecture#1 and the discussions on NYU Brightspace.