Systems Programming — Università della Svizzera italiana
A. Carzaniga, E. Fynn, P. Coelho

Graded Assignment n. 2
May 2, 2018

Graded Assignment 2: Virtual File System

Due date: May 11, 2018 at 20:00

This is an individual assignment. You may discuss it with others, but your formulations, your code, and all the required material must be written on your own. In any case, you must acknowledge the sources used and clearly mention any help received from colleagues.

1 Problem

Virtual File System (VFS) is an abstraction layer on top of a more concrete file system. 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. It can be used to bridge the differences in Windows and Unix filesystems, so that applications can access files on local file systems of those types without having to know what type of file system they are accessing [1].

Your task is to provide a VFS library that allows operations like create, open, write and read on virtual files that can be stored either in memory or in a actual filesystems on disk.

2 Implementation

Your code should implement the following header and applications should use it as a shared library.

```c
#ifndef VFS_H_INCLUDED
#define VFS_H_INCLUDED
#include <stdlib.h>

struct vfs;
struct vfile;

enum vfs_type {VFS_MEMORY, VFS_DISK};

struct vfs *vfs_open(enum vfs_type t, const char* root_folder, size_t folder_len);
int vfs_mkdir(struct vfs* root, const char *path, size_t path_len);
void vfs_close(struct vfs *root);
struct vfile *vfile_open(struct vfs* root, const char *file_name, size_t name_len);
int vfile_write(struct vfile *f, const char *data, size_t data_len);
int vfile_append(struct vfile *f, const char *data, size_t data_len);
size_t vfile_read(struct vfile* f, const char *data, size_t data_len);
void vfile_close(struct vfile *f);

#endif //VFS_H_INCLUDED
```

In this header, `struct vfs` is a opaque data type that represents the VFS and that should hold all the necessary metadata. The functions related to this virtual file system receive the object representing the VFS as parameter:

- `vfs_open()`: creates a new virtual file system whose root folder and type are given as parameters. The root folder must be a valid absolute path if the type is `VFS_DISK`. A `VFS_MEMORY`-type virtual file system holds the directory structure inside in the main memory as a tree (see Section 2.1

for details), while a `VFS_DISK`-type one creates the root folder on disk and consider every further command to be relative to such folder. A `NULL`pointer is returned if any error occurs.

- `vfs_close()`: closes the virtual file system and frees all associated memory. In-memory files and folders should be destroyed. On-disk files and folders should be kept.
- `vfs_mkdir()`: creates the specified directory specified by the relative path (from the root folder) and returns 1 in case of success or 0 otherwise.

In addition to a virtual file system, the library also provides *virtual files*, represented by `struct vfile` objects. Virtual files are stored in folders inside the virtual file system. An application can open/create, read, write, and close a virtual file. The detailed description of each function associated with virtual files are as follows:

- `vfile_open()`: Opens the specified file or creates an empty one if the containing folder in the virtual file system exists. It returns the pointer to the virtual file or `NULL`otherwise.
- `vfile_write()`: Writes data from the beginning of the virtual file, and returns 1 in case of success, or 0 if the virtual file does not exist or other problem occurs.
- `vfile_append()`: Writes data from the end of the virtual file, and returns 1 in case of success, or 0 if the virtual file does not exist or other problem occurs. Successive "appends" to the same file should keep adding data to the end of the virtual file.
- `vfile_read()`: Reads up to `data_len` bytes from virtual file and puts into informed buffer, returning the number of bytes actually read.
- `vfile_close()`: Closes the virtual file and frees all associated memory. The content of in-memory virtual files should be kept until the virtual file system is closed. On-disk files content are always kept.

Successive calls to `vfile_read`, `vfile_write`, and `vfile_append` on the same virtual file should continue from where previous call left.

2.1 In-memory tree

When you open a new in-memory virtual file system, the library creates a new tree with a single node (the *root*), and the "folder name" information, as shown in Figure 1.

name : /home/carzaniga

Figure 1: Tree after a call to *vfs_open(VFS_MEMORY, "/home/carzaniga", 16).*

Now suppose you invoke the following functions in you library:

```
vfs_mkdir(root, "courses/sysprog", 17);
vfs_mkdir(root, "vacations/", 12);
f = vfile_open("vacations/address.txt", 22);
vfile_write(f, "Via G. Buffi, 13", 17);
```

Your tree should now be like Figure 2. Inner nodes are always folders, while leaves may be either a file or an empty folder. In this example the gray nodes are folders and the green one is the file we have created with `vfile_write()`.

Your library should manage such tree, taking care of allocating and freeing memory as needed to keep a consistent representation of the folders structure in the tree.

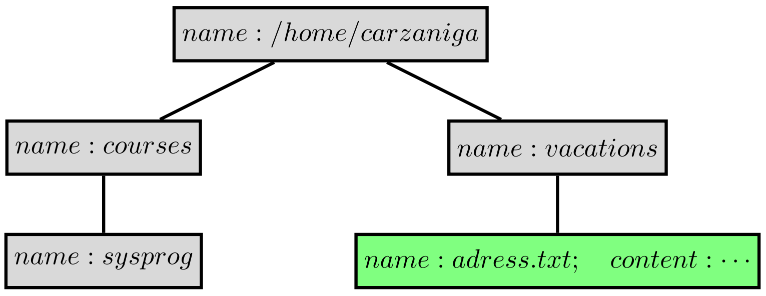

Figure 2: Tree after vfs functions calls.

2.2 On-disk tree

The underlying on-disk file system (ext4, ntfs, exfat, . . .) takes care of the metadata associated with folder and files. Thus, your implementation should invoke the corresponding system calls to keep the folder structure on disk consistent with the virtual file system version. For instance, a call to `vfs_open (VFS_DISK, "/home/carzaniza/myfiles")` should create the corresponding folder, i.e. , if you run “ls” in the folder “/home/carzaniga” there should exist a folder named “myfiles”.

Similarly, subsequent calls to:

```
vfs_mkdir(root, "courses/sysprog", 17);
vfs_mkdir(root, "vacations/", 12);
f = vfile_open("vacations/address.txt", 22);
vfile_write(f, "Via G. Buffi, 13", 17);
```

should create folders “courses”, “courses/sysprog” and “vacations” inside “/home/carzaniga/myfiles”. A file “address.txt” should also be inside the folder “vacations”.

3 Compiling the library

Other applications must link their own code to your library in order to use it. To build your code as a shared library, please read carefully the documentation at `http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html`. Pay especial attention to section 3.4.

4 Test cases

Along with this document we provide some test cases you might run to test your implementation. A failure to run or pass the test cases means your library is not implemented according to the specification.

References

[1] Virtual File System wikipedia: `https://en.wikipedia.org/wiki/Virtual_file_system`