Operating Systems, CSE531

Assignment 6: MemoryFS

Due Date: Nov 12, 2013, 1700 Hrs

1 Objectives

- Learn basics of VFS APIs, especially about FUSE, Filesystem in USErspace.
- Understand the typical attributes of a filesystem in memory, and a naive implementation of them.
- Mapping to the desirable virtual memory areas for this FS. For eg. Swap space or Ram space

2 Introduction

In this assignment, you will develop two *read-only/read-write* file system drivers for the MemoryFS file system. In short, you have to mount a read-only FS at one directory and a read-write FS at another directory such that all files in mounted space does only exist in primary memory. You can derive the ideas of MemoryFS file system from already existing **tempFS, ramFS** filesystems which stores files in volatile space of RAM.

There exist as many file system APIs (the so-called VFS API) as there are different operating system kernels. The FUSE (File system in User space) API is another file system API, however it allows the development and execution of file system drivers in user land instead. The advantage of this approach is that less code is run in the kernel, which reduces the risk of crashes or exploits. Since FUSE presents a simple, abstracted file system interface, it is easily possible to use FUSE file system applications on different operating systems, such as Linux, FreeBSD, Mac OS, OpenSolaris or even Windows. Running in user land also enables the use of various language bindings, allowing the implementation of FUSE file systems in different programming languages such as C, C++, Java, Python, Perl, etc.

3 Details

Read-only file system driver for MemoryFS

1. Implementation should be able to deal with long file names
2. It has to support the getattr, readdir, statfs and read FUSE file system operations
3. You can use the sample code Provided in Courses portal

Read-Write file system driver for MemoryFS

1. Impementation should be able to deal with long file names
2. It has to support following operations
    - Directory operations: readdir,mkdir,rmdir
    - File operations: open,read,write,truncate,access,close
    - Meta operations: getattr,statfs,symlink,link,unlink,rename etc.
3. You should provide an option before mounting FS that whether it is to use RAM or use Swap space.

4 Running and Testing

Development environment

To build the code, you will need the FUSE library in user land. On a Linux distribution that supports deb packages, such as Debian and Ubuntu, you can install it by the following command:

```
% sudo apt-get install libfuse2 libfuse-dev
```

Mounting

The supplied skeleton already performs basic FUSE argument parsing. To mount the file system, run the application with file system and destination mount point as arguments:

```
% mkdir tmp
% ./MemoryFS_r -f tmp | ./MemoryFS_rw -f tmp -usage swap/ram
% ls -l tmp
```

P.S: above MemoryFS r and MemoryFS rw are name of executable, and -f option allows fuse to stay in foreground, which is better for debugging. -usage option is for which VMA is to be used.

Unmounting

In case your file system does not shut down properly, you might experience an error message, such as *ls: cannot access tmp: Transport endpoint is not connected.* In this case you will have to unmount the file system manually:

```
% fusermount -u -z tmp
```

Testing

We provide a compressed sample file to test, which will unzip in MemoryFS, and also we will run dd to allocate files of bigger sizes to check how you handle swap and RAM. So a testing file can go from 1MB to 1 GB. You can test your implementation with standard mount by comparing the structure of the mounted sample file system. You are also encouraged to use your custom designed shell to run the commands in the given filesystem space. Try out basic commands like cat, ls, echo etc. and most of the time these commands should work

5 Deliverables

- Full source code for the assignments
- Short project report about your experiences on implementation and testing (no more than two pages)

Guidelines

1. *All italics, bold and color fonts are important, so please adhere to them*
2. **Indent your codes properly, comment wherever needed and try to organize code in form of modules.**
3. Your code should compile and run correctly at time of evaluation.
4. Deadline is strictly 5*:00 P:M*. No further delays, and submission via mail will not be considered.
5. Create folder named you roll number, put source files and report in it .. Tar it (name tar as Assignment6.tar.gz) and *upload.*