Cameleonica: Conceptual Design

Functional and quality requirements for acceptable usability

Arkadiusz Bulski

Abstract

This paper describes a conceptual design of the project, that is the functionality, the semantics, the characteristics, and the quality that will be offered to end users. Rationale for some fundamental decisions is explained. Operational scenarios are presented, followed with acceptance tests to establish an acceptable level of usability.

Introduction

Cameleonica project is an innovative filesystem design. Both new commands and well established commands with new semantics are given into hands of end users. Both are highly specialized and broad in application. Consider following operations.

Commiting and reverting history of changes is typical of version control systems, with Git for example. Encrypting entire disks is quite common, with TrueCrypt. Hiding files and even entire systems on disk is possible, again with TrueCrypt. Compressing and hashing files is a standard operation, with built-in utilities. They are all available separately, and furthermore, they are all implemented outside of any filesystem. This approach is good for simplicity, as underlying filesystems need to implement only very basic operations. However, an opportunity is missed this way. Filesystem is put into a position where it can provide guarantees that are not achievable to unprivileged programs having no access to underlying storage device, or precompute/cache data for more than one application in a trusted manner. This project aims to offer exactly that.

Whole project can be defined in few (somehow separate) parts: backend system calls through which user processes can access content, their semantics, frontend command-line and graphical tools with which users can manage entire filesystem, on-disk format to which other implementations can refer to, standards compliance, etc.

System call interface

Processes gain access to files through a long existed mechanism called syscalls. For any filesystem to be accessible, certain syscalls need to be implemented. POSIX and Linux standards specify the details. We shall skip the specifics of what exactly needs to be implemented and how, and generally say that following syscalls will need to be provided:

> *open, creat, read, write, pread, pwrite, readv, writev, lseek, dup, close, chown, chmod, fallocate, ftruncate, utime, link, symlink, unlink, mkdir, rmdir, rename, fsync, fdatasync, fsyncdir, sync, access, stat, statfs, fcntl, ioctl, flock, getdents, listxattr, getxattr, setxattr, removexattr, quotactl, getrlimit, setrlimit*

List above may not be accurate. Some syscalls may be removed or added later on. The list only provides a basis for further, more elaborate evaluation of what needs to be provided to meet expectations of super users. Other, modern, popular filesystems may be used to define how these syscalls should work, in addition to on-paper specifications.

Above that, syscalls may be subject to semantics related to specialized features that are Cameleonica-specific and have no counterpart in other filesystems. How operations are affected by these semantics will be defined in further sections, with regard to filesystem as a whole rather than single operations.

Command-line interface

Syscalls can be used to access files within a mounted filesystem but the filesystem must first be created with command-line utilities. Further management of the filesystem would also be done using utilities. Following operations should be available:

- ✔ initializing a filesystem within a container file (regular file or block device)
- ✔ configuring off-line changeable volume settings (masterkeys)
- ✔ mounting a possibly interrupted filesystem
- ✔ configuring on-line changeable file settings (auto-compression)
- ✔ checking integrity of all internal data-structures
- ✔ defragmentation of viable data-structures
- ✔ compression of viable data-structures
- ✔ management of directory-tree snapshots
- ✔ serialization into a compact incremental backup
- ✔ resizing and moving underlying partition on-line
- ✔ emergency recovery of a broken filesystem
- ✔ destruction of a decommissioned filesystem

Graphical user interface