Computer System Engineering

Design Project 1 Report

# Create a Versioning File System

*Author:*
Bo Song 11302010003
YiTing Cheng 11302010050
YuWei Zhou 11302010067

April 5, 2013

## 1 Introduction

The goal of this design project is to create a versioning file system. The versioning or continuous snapshotting file system is defined to store all versions of each file over time. When user modifies a file or a directory, the versioning file system create a new version after closing the file or directory and store the old version which can be only read by users.

To implement this system, The whole design will solve two major problem: how to maintain the old version and how to manipulate them, as well as achieving low access time and better space utilization under common use cases.

## 2 Design Description

The versioning file system(VFS) represents different versions of a file or directory as different inodes which is based on the inodes in Unix file system. Therefore, we need add extra fields to origin inode data structure. Then, a new layout and management is required to maintain the huge number of inodes and data blocks. Finally, some interfaces is adopted to let users manipulate different versions easily.

### 2.1 Data Structure

There are several fields are added to both the on-disk and in-memory representation of the inode as mentioned below. Since the size of inode is fixed(typically 2K in fast file system), the number of indirect block pointer fields will decrease after adding new fields and it will result in reduced max file size under approximately 10%.

**Timestamp** When a new version is created, a unsigned integer is recorded, representing the number of seconds passed since a epoch which is predefined in the system. If the unsigned integer is 32 bit, the timestamp allows the system to represent about 132 years after the epoch(typically 1970.1.1) which is enough to current system, and it is easy to extend to 64 bit when we need to represent more time.

**Bitmap** The system embeds bitmaps in inodes and indirect blocks that allow system to record which blocks have had a copy-on-write performed which will discuss in memory management section. A bit of value 0 indicates that a new block needs to be allocated on the next write and bit value 1 indicates that a new allocation of this block has been performed.

**Next** A pointer to the next old version of an inode. Through this field, we can search a specific version of an inode like an linked list, and the current version inode is the head of linked list. Since the performance of it may not so considerable, a optimized structure will discuss in layout section.

**Record** When a user want excludes a file or directory from versioning, an extra bit is employed to indicate it. a bit of value 0 is versioning which is default, and one is not versioning. When the system wants to create a new version, it will first exam this bit and decide whether to do subsequent things.

**Head Pointer**

## 2.2 Memory Management

When a new version is created, it is too waste to copy all the block in the old version. So copy-on-write(abbreviated COW) is adopted to implement multiple versions of data compactly. The system needs to create a new physical version of a file only when data changes. Frequently, physical versions have much data in common. The COW allows versions to share a single copy of file system blocks for common data and have their own copy of data that have changed. As a result, it extremely improves the memory utilization in the system.

How COW works is showed in the following steps:

## 3 Analysis

## 4 Conclusion

## 5 Acknowledgment