Computer System Engineering

Design Project 1 Proposal

Create a Versioning File System

Author:
Bo Song 11302010003
YiTing Cheng 11302010050
YuWei Zhou 11302010067

April 5, 2013

Overview

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(specifically, write or truncate something in a file) or modifies a directory(add, delete or rename files in it), 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. The system can be realized by extending inodes, interfaces and adopting copy-on-write memory management strategy based on Unix file system.

Design Description

Data Structure

To implement different versions of a file, some extra fields are added to the inode structure as follows.

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.

Bitmap - Maintaining the block versioning state of file or directory, in order to realize COW memory management, checking whether a new version is needed and performance optimization.

Next - A pointer to the next old version of an inode.

Record - A bit to indicate whether an user excludes the file or directory from versioning. Zero is default which is not excluded.

Head Pointer - A pointer to a block containing addresses of the head of sublists of inodes discussed in layout section.

Memory Management

When a new version is created, it is 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. When we open a file, we duplicate the inode first. The duplicated inodes initially share all data blocks(including direct and indirect blocks) in common. When a block is update, which means writing or truncating in a file or add or delete files in a directory, the file system allocates a new physical block to hold the new data and change the block address and the related bitmap in the inode. When the user close the file, the new version of inode is finally created with only a few extra blocks. If there is no modification, which means bitmap is all zero, the duplicated inode will release. The COW strategy extremely improve the memory utilization in the system.

Layout

The problem that the length of a linked inodes list may be very long will cost more time to find a specific old version. Hence, a optimized layout is introduced to solve this problem. When the length of linked list exceeds a predefined number, the system divides it into small sublists. The head of each sublist which contains the address of first inode in the sublist and its timestamp is recorded in a block mentioned in data structure section(Figure 2). When the user find a version of file with given timestamp, the system use the timestamp as a key and search the sublist containing that inode efficiently with a function.
When the disk is almost full, the system also support garbage collection which frees the oldest sublist.

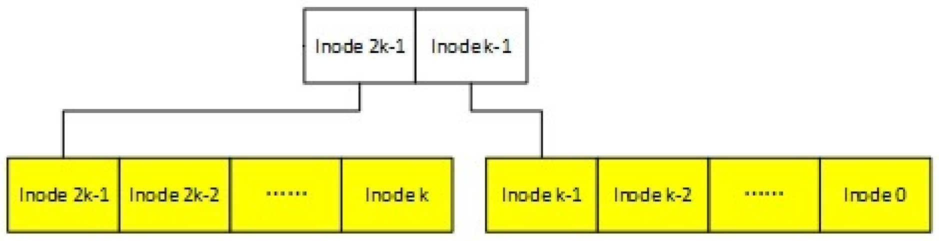

Figure 1: Layout
Constant k is predefined max length of sublist. White block is head block reference and yellow block is linked sublist inode block.

Manipulating Old Versions

User can access any version of file or directories by appending @ and timestamp, such as “cd \home@1032433”. When the system resolve the file or directory names, it

reads from right to left and regard the last @ and number as version specifier. Hence, it can distinguish the version specifier from @ and numbers in its origin name.
Since the timestamp is not user-friendly, it has two extra ways to get the version we want.The first one is to lookup the log file or use list_version system call to enumerate all the version of a file or directory.The other approach is use a inaccurate timestamp and the layout in previous section will choose a sublists and result in a nearest version.

Conclusion

The above design meets the requirements as outlined in the problem statement by strengthening inode structure. Although we employ copy-on-write strategy to reduce a lot of space cost, the system tends to sacrifice space for time such as bitmap in each inode and extra blocks used in the file system layout. Future questions that still remain are the specifics of the implementation, as well as a better layout algorithm and use case analysis.