An LFS cleaner framework for Linux and other UNIX-like operating systems

30th November 2001

Dwight Tuinstra (tuinstra@clarkson.edu)
Clarkson University
Potsdam, New York 13699

Category of Project: File System Improvements

Background

Log-structured filesystems (or “LFSes”) [5, 8] preserve the inode semantics of traditional file systems, but take a radical approach to placement of data on the drive. Data and metadata are written sequentially to a log. The log is not a staging area for later insertion into stable store, but rather is the stable store itself. This avoids the high costs of seeks suffered by file systems that attempt to carefully place data onto the disk. Additionally, a log structure enables development of features such as rapid crash recovery, file system cloning, and file “undelete” operations.

LFSes also write updates to the head of log rather than performing an update-in-place. The position of blocks containing the updated data is noted in the log, implicitly invalidating the old blocks. A form of garbage collection is needed to reclaim space taken by invalidated blocks. This is performed by a user-space utility called a “cleaner”. Design and implementation of cleaners is one of the primary challenges in creating an efficient and robust LFS.

At present, there appears to be only one complete and working open-source LFS implementation: that of the NetBSD operating system [7]. Linux has the LinLog LFS effort [1, 2], but it lacks a fully operational cleaner. Initial work on a cleaner has been done[3], however it is preliminary and development appears stalled. The Linux-based Swarm/Sting project was described in 1999 [6], but a public release of the code seems not to have happened yet.

Part 1: Project and Objectives

This project seeks to provide a robust, well-documented, fully-functional cleaner for the LinLog LFS; and in the process produce a general framework for constructing LFS cleaners.
Part 2: Methodology and Design

Rather than push forward on a preliminary and unproven native Linux cleaner, the approach is to generalize the NetBSD cleaner into an LFS cleaner framework, then specialize that framework for the Linux LinLog LFS. The general framework will be released under a BSD-style license since it is derived from BSD-licensed code and will be donated back to the NetBSD project. Linux-specific portions will be released under the GPL. The BSD license is an open source license.

Language Any kernel components will be written in C. C++ will be used for the rest, for its support of object-oriented programming, improved error-handling through exceptions, and better memory management through constructors and destructors (cleaners make heavy use of scratch memory). The improved modularity and facilitation of code refactoring should enable implementation of LFS enhancements proposed in the literature [4].

Project Phases (1) Refactor the existing NetBSD cleaner to simplify it and split it into general and NetBSD-specific components. (2) Release/donation of mid-beta-quality refactored code to the NetBSD project. Anticipated date: January 2002. (3) Implement the specialization classes for LinLog. Adjust the framework as necessary to tease out true generalizations, back-porting these to the NetBSD cleaner. (4) Release/donation of mid-beta-quality code to the LinLog project. Anticipated date: February/March 2002. (5) Develop kernel portions of LinLog to support improved performance (such as efficient provision of a “seiguse table” as mentioned below). Anticipated start date: March 2002.

Design The design models the high-level objects in an LFS, encapsulating the implementation details of a specific LFS. The general LFS cleaner acts on these objects. Major classes are as follows:

CleanerCtl: models all control parameters that affect the behavior of the cleaner, such as timeout periods, amount of cleaning per cleaner wakeup, debug and logging levels, and so on.

LFSInfo: models high-level information about a mounted LFS.

InodeMap: Since LFSes write modified inodes to the head of the log, there must be a means of discovering the on-disk location of a given inode. The InodeMap models the LFS implementation structure that contains this mapping.

Segment: Writing and cleaning in an LFS takes place at the level of “segments”, which contain data blocks and inodes from (possibly multiple) files. The LFS on-disk partition is modelled as an array of Segments.

SeguseTable: This is a fast-access summary of usage and aging statistics for each segment, updated dynamically by the LFS kernel components. The cleaner reads this information to decide which segments to clean. (The LinLog LFS lacks this feature, making any cleaning extremely inefficient. Until this feature can be added to LinLog’s kernel components, the LinLog implementation of this class will require accessing the actual on-disk segments.)
Part 3: Results

The project is presently in phase one, abstracting and generalizing the NetBSD cleaner code. This is a non-trivial task due to layers of maintenance and features added to the original 1992 BSD 4.4 code [7]. Current work, including UML diagrams, can be found at [10]. On a previous refactoring attempt (prior to the goal of constructing a general framework), approximately half of each of the two major cleaner code files (cleanerd.c and library.c) had been reconstructed in an object-oriented fashion and were running in regular use on the author's workstation.

References


