The Logical Disk User Documentation

Robert Grimm, Wilson C. Hsieh, and M. Frans Kaashoek

Laboratory for Computer Science
M.I.T.
Cambridge, U.S.A.

E-mail: `ld-info@pdos.lcs.mit.edu`
WWW: `http://www.pdos.lcs.mit.edu/ld/`

1 December 1995

1 Introduction

This document gives a short introduction to installing and compiling the log-structured prototype (LLD) of the Logical Disk (LD). The Logical Disk defines a new interface to disk storage that separates file management and disk management. The LD interface is designed to support multiple file systems and to allow multiple implementations, both of which are important given the increasing use of kernels that support multiple operating system personalities. The log-structured prototype of LD is based on the assumption that most reads are absorbed by the file system cache and that therefore disk traffic is dominated by writes. LLD writes dirty file system blocks to disk in long contiguous writes. It divides the disk into large, fixed-size segments; the segment being filled is maintained in main memory and written in a single disk operation.

We have implemented LLD as a user-level process and linked it to the Minix file system (MinixLLD). MinixLLD can use a file or a raw disk partition to store data. It runs on Sun workstations running SunOS 4.1.3, but we expect that LLD will run on most BSD-like operating systems with minor changes.

MinixLLD is intended as a file system for experimentation and evaluating research questions. It is *not* ready to be used as a production file system. We can not make any guarantees in any way.

The following research papers discuss the Logical Disk, the log-structured prototype and atomic recovery units (a mechanism that allows several disk operations to be executed as a single atomic unit with respect to failures):

- Wiebren de Jonge, M. Frans Kaashoek, and Wilson C. Hsieh. The Logical Disk: A New Approach to Improving File Systems. In *Proceedings of the 14th Symposium on Operating Systems Principles*, pages 15–28, 1993.
- Robert Grimm, Wiebren de Jonge, Wilson C. Hsieh, M. Frans Kaashoek. Atomic Recovery Units: Failure Atomicity for Logical Disks. To be published.

The rest of this document is structured as follows: Section 2 gives an overview over the source tree and highlights some important compiler flags and source files. Section 3 describes the steps necessary to compile LLD. Section 4 describes how to run LLD. This document assumes that the source distribution for the Logical Disk has been downloaded, uncompressed (using `gunzip ld-1.0.tar.gz`) and untared (using `tar xf ld-1.0.tar`). All directory references in this document are relative to the top-level directory for the source distribution. The source distribution and related documents can be accessed through the world-wide web at `http://www.pdos.lcs.mit.edu/ld/`.

2 Setting up LD

The source distribution of the Logical Disk system is organized as follows:

Directory	Content
L_disk	LD sources for the old version (with sequential ARUs).
LLD_with_ARUs	LD sources for the new version (with concurrent ARUs). This version also features several bug fixes over the old version.
fs	MINIX file system modified to run with LD.
include	Various include files.
command	Code to build an initial file system, notably `mk_lld`.
unix	Code to hook up MinixLLD and an interactive command shell via UNIX pipes.
doc	This user documentation.

Both the code for the Logical Disk and the Minix file system are linked into one executable to produce the MinixLLD system. The MinixLLD system can be run in two ways, first, by itself and, second, using an interactive shell. To run the MinixLLD system by itself the `EXPLICIT_WORK` flag needs to be defined during compilation (the `Makefile`s in the `fs` and the LLD source directories need to be changed). The function `do_experiment()` defines the operations that are to be executed when running MinixLLD by itself. It is defined in the file `fs/performance.c`, which also contains several examples of how to write `do_experiment()` (and illustrates the use of the compiler flags of the form `[`*name*`]_EXPERIMENT`).

To run the MinixLLD with the interactive shell we use the two helper programs `startup` and `rfs`. `startup` takes two programs as arguments, creates two pipes (one for each direction of communication between the two programs), forks itself and then runs the two programs. `rfs` provides the interactive shell, reading commands from the console, sending them to MinixLLD and then printing the returned results. The source code for both helper programs can be found in the `unix` directory. MinixLLD must be compiled without the `EXPLICIT_WORK` flag to be used with the interactive shell. The commands of the interactive shell are defined in a table in the file `unix/rfs.c`.

MinixLLD uses a raw UNIX partition for creating and accessing its file system. The name of this partition is defined in `L_DISK_NAME` in the file `disk.h` in both the `L_DISK` and the `LLD_with_ARUs` directories. It is also defined in `L_DISK_NAME` in the files `readblock.c` and `readdisk.c` in the `unix` directory, which define helper programs to read data from the MinixLLD file system within UNIX. The partition is prepared for use by MinixLLD by running the `mk_lld` command in the `command` directory. **Note that `mk_lld` will erase all previous data in the MinixLLD partition.**

The on-disk data structures for the new version of LLD needed to be changed (expanded) from the old version of LLD (to accommodate concurrent atomic recovery units). However, to use both versions of LLD on the same file system (modulo concurrent atomic recovery units) and, more importantly, to use the same `mk_lld` command for both versions, the flag `VERSION_MERGER` must be defined during compilation.

3 Compiling LD

The following steps describe how to compile the new version of LLD. Since the `mk_lld` command links in some of the object files also used in the old version of LLD, the old version must also be compiled (using the `VERSION_MERGER` flag).

1. Set up the source distribution of the Logical Disk

2. Make sure that `L_DISK_NAME` is defined correctly (in the file `disk.h` in both LLD source directories).
3. Depending on which way you want to run MinixLLD, define (or undefine) `EXPLICIT_WORK`. If you use the `EXPLICIT_WORK` flag, also define your `do_experiment()` function and the corresponding flag of the form `[`*name*`]_EXPERIMENT`. The `EXPLICIT_WORK` and the `[`*name*`]_EXPERIMENT` flags are defined in the `Makefile` of both the Minix and the LLD directories.
4. Make sure that the `VERSION_MERGER` flag is defined in the `Makefile` of the LLD source directories and the `command` directory.
5. Compile the Minix file system by issuing `make fs` in the `fs` directory. Ignore the linker error messages.
6. Compile the MinixLLD system by issuing `make lld` in either of the two LLD source directories. You will need to make the old version of LLD for the next step.
7. Make the `mk_lld` command by issuing `make mk_lld` in the `command` directory.
8. You are all set to run MinixLLD by itself now. If you want to use the interactive shell, go on with the next two points.
9. Make the `startup` command by issuing `make startup` in the `unix` directory.
10. Make the `rfs` command by issuing `make rfs` in the `unix` directory.

4 Running LD

A fresh MinixLLD partition is created by the following command (to be executed in the `command` directory):

`mk_lld` [*Device*] [*Number of Blocks*]

For example, to create a 400 MByte partition (the standard block size is 4 KByte) on disk `rsd1d` the following command would be issued:

`mk_lld /dev/rsd1d 100000`

There are two ways of running LD: First, the experiment can be linked in with the MinixLLD executable. In this case, it is sufficient to simply execute the MinixLLD executable. Second, to run LD interactively the following command needs to be issued in the `unix` directory:

`./startup` [*LD executable*] `rfs`

For example, to run the standard LD executable (with concurrent ARUs) the following command would be issued:

`./startup ../LLD_with_ARUs/lld rfs`

The individual commands of the interactive shell are listed in a table in the source file for `rfs` (`unix/rfs.c`). Note that superuser privileges are necessary for above commands to work.

5 Copyright Notice

The bulk of the LD distribution (both source code and supporting documentation) is covered by the following copyright:

Some parts of the LD distribution (notably the Minix sources, see the source distribution) are covered by other copyright holders/policies.