Reprinted from the

Proceedings of the Ottawa Linux Symposium

June 26th–29th, 2002
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Wild Open Source, Inc.*

Reverse engineering an advanced filesystem

Christoph Hellwig
LST e.V.
hch@lst.de
http://verein.lst.de/~hch/

Abstract

The *VxFS* filesystem from VERITAS is an example of a UNIX filesystem that not only offers a broad range of advanced functionality, such as extent based allocation, intent logging, snapshoting, but also has on-disk formats with various differences between the versions and ports.

FreeVxFS is a freely available implementation of the VxFS filesystem format for Linux, implemented only by looking at the very rarely available public documentation and reverse engineering the SCO UnixWare version of the commercially available VXFS driver.

1 Introduction

Today's operating systems feature a wide vary of commercially available advanced filesystems. Only for a small number of such filesystems (for examples IBM's *JFS2* family or SGI's *XFS*) does a freely available Linux kernel implementation exist.

Linux contributors already have implemented support for many simpler proprietary filesystems like *FAT* or *EFS*, but the there are only few independent implementations of complex filesystem designs such as Microsoft's *NTFS*.

The VERITAS Filesystem (VxFS) originated from a joint-venture of VERITAS and the Unix System Laboratories (USL) in the early 90's of the last century to develop an advanced filesystem for System V Release 4 Unix (SVR4), featuring capabilities such as read-ahead logging (commonly called journaling) and copy-on-write snapshots at the filesystem level. Today it has been ported to a great number of Unix derivatives such as Sunsoft Solaris, Sequent (IBM) Dynix/ptx, Hewlet-Packard HP-UX, or Caldera OpenUnix; and non-Unix platforms like Microsoft Windows 2000.

VxFS is a proprietary VERITAS product and delivered in source or binary form to paying customers. There is, on the other hand, no public documentation of the on-disk format used by VxFS, which makes implementations other then VERITAS' difficult to implement. There is need to access VxFS-formated disks from Linux for various reasons, like migration from obsolete Unix platforms or access to foreign files for development. (In the author's case this was the Linux-ABI binary emulation framework).

In the first Quarter 2002, VERITAS announced the commercial availability of the VERITAS

Foundation Suite for Linux, featuring a port of their VxFS-implementation. Being tied to obsolete versions of the Red Hat kernel package and priced in 4-digit US-dollar range, it is not an option for most possible FreeVxFS uses, though.

2 Legal Background

When implementing a non-trivial piece of software that inter-operates with another software you either need a written specification of the interface, or you need to look at the other software, using helper tools such as disassemblers; this is called *reverse engineering*. In the VxFS case there is no proper documentation of the filesystem's layout as stored on disk so *reverse engineering* is the only available choice. In the European Union reverse engineering of software products is handled by the Directive on Software Copyright Protection from 14 May 1991. In Article 6 (Decompilation) it states the following:

> The authorization of the rightholder shall not be required where reproduction of the code and translation of its form within the meaning of Article 4 (a) and (b) are indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs, provided that the following conditions are met:...

As the creation of a Linux driver for a proprietary filesystem matches the terms of interoperability used in this EU law exactly, we are explicitly allowed to examine an existing, licensed installation of VxFS to gain information for implementing a free replacement for Linux.

Another interesting legal problem came up when the released driver was merged into the official Linux kernel tree, as VERITAS claimed the use of "vxfs" as driver name was threatening their Trademark. The driver name was changed to freevxfs to avoid further legal problems.

3 Getting Started

Structure is more important than code—this golden programming rule is even more important when trying to archive format compatibility with an existing software.

To implement an independent driver for a filesystem layout, one needs to know every single structure that is written on disk in detail to archive full compatibility. On the other hand the inner workings of the drivers might be completely different, especially if they were written for different environments (e.g. operating systems in this case).

Thus the first step to produce a free VERITAS filesystem driver for Linux was to create a full description of the disk layout that is not directly derived from the original code. As starting point I used the public available documentation of UNIX vendors shipping VxFS with their products, but the results were discouraging, there were only two documentation sets that contain non trivial information about the VxFS disk on-disk format: first the VxFS System Administrator's [AdmGuide] has a chapter titled "VxFS disk layout" which contains a very high-level documentation of the basic format elements; second the inode_vxfs (4) [Inode] and fs_vxfs (4) [Fs] on HP-UX contain

a description of many fields of the VxFS superblock and inode, but neither document defines even one element of the VxFS structure completely.

To get a suitable description despite this lack of human readable-format description, reverse engineering techniques had to be applied. There are three major reverse engineering procedures in the context of software: disassembly, use of symbolic debugging information, and protocol snooping. For the development of FreeVxFS, only the first two were used, as protocol snooping of block storage devices requires special and only rarely available hardware (with hardware-emulators like Bochs this is in the process of becoming easier).

4 Symbolic Debugging

Any modern C compiler has a mode to include information about the source-level representation with a binary program to allow high-level debugging with tools like GNU gdb.

VERITAS' VxFS product does not contain any program with such debug information, but it contains C headers files to allow access to its structures from custom programs.

Technically these files could be directly used by a free implementation of VxFS to use its definition similar to free programs using other system headers on proprietary operating systems. The problem with this approach is that it requires the compilation of the driver to happen only on systems with licensed installations of VERITAS' filesystem product, thus disallowing a free operating system to be used as development host.

So instead of directly using the header files, the structural information is extracted from the them using the aforementioned debugging methods to create a set of new and entirely free headers that define the VxFS on-disk format. The program that pulls in the headers is very simple, as it needs to have no functionality at all—it exists only to allow debugging information to be generated.

```
#include <sys/types.h>
#include <sys/time.h>

/* fix compilation with gcc */
#define uint8_t __junk

#include <sys/fs/vx_machdep.h>
#include <sys/fs/vx_gemini.h>
#include <sys/fs/vx_param.h>
#include <sys/fs/vx_layout.h>

main()
{
}
```

Once compiled with debugging options enabled (e.g. `gcc -g`), we have a binary suitable for attacking with a debugger such as gdb. This process is time-consuming as the names of the different record types have to be guessed from the previously mentioned public documentation and often one of the custom types embeds a number of other such types. In addition all scalar types are shown in form of C language basic types by gdb and all typedef information is lost. To allow a portable format description that can also be used (e.g. on computers with 64-bit wide longwords) all occurrences of types that have different sizes on different Linux ports must be replaced with proper, explicitly sized types. The following example gdb session shows the definition of the on-disk inode used by VxFS (`struct vx_dinode`):

```
(gdb) ptype struct vx_dinode
```

```
type = struct vx_dinode {
    struct vx_icommon di_ic;
}
(gdb) ptype struct vx_icommon
type = struct vx_icommon {
    long int ic_mode;
    long int ic_nlink;
    long int ic_uid;
    long int ic_gid;
    vxhyper_t ic_size;
    struct timeval ic_atime;
    struct timeval ic_mtime;
    struct timeval ic_ctime;
    char ic_aflags;
    char ic_orgtype;
    u_short ic_eopflags;
    long int ic_eopdata;
    union vx_ftarea ic_ftarea;
    long int ic_blocks;
    long int ic_gen;
    vxhyper_t ic_vversion;
    union vx_org ic_org;
    long int ic_iattrino;
}
(gdb)
```

The symbolic debugging information was the most useful resource during the FreeVxFS development.

5 Disassembly

The author has examined most of the UnixWare VxFS binary driver module with various disassemblers.

The simplest form of disassembly can be produced by the program `objdump` from the GNU binutils package. Its output for trivial vnode operations appears in Figure 1.

Commercially available disassemblers like DataRescue's IDA Pro (the windows version runs under Linux/wine) offer additional features such as the naming of data types or additional symbol resolving that should not be discussed further here.

Although this uncovered a number of interesting facts like the exorbitant stack usage in VERITAS's driver, the disassembly of the UnixWare VxFS driver did not uncover a notable amount of information useful for the current publicly available read-only FreeVxFS versions. On the other hand, the slowly progressing development of write support would be impossible without the use of disassembly, mostly because a number of bitmap operations in the block/inode allocators is not documented in any other way than the program itself.

6 Implementation

As already mentioned previously, FreeVxFS targets the Linux kernel from version 2.4 upwards. There are various reasons for this choice:

- Linux is the free operating system with the biggest overall user count. The 2.4 kernel was the upcoming stable release when the development was started.
- The Linux kernel allows runtime loading of independently developed filesystem modules. This allowed the early FreeVxFS development to happen without ties to the direct kernel development and with very short turnaround times.
- All recent Linux kernels feature a rich set of generic routines that can be used in filesystem code. Starting with the 2.4 kernel most of the data I/O path is handled by such generic code, thus letting filesystem developers concentrate on difficult aspects of their particular filesystem implementation.

The early FreeVxFS development was done as a project separate from the main Linux kernel tree and supported different kernel versions with the same codebase. Starting with the 2.4.6 prereleases it merged into the official Linux kernel tree and is maintained as part of it.

Like most filesystem drivers, FreeVxFS development was done incrementally in the beginning—that means support for the different parts of the on-disk format was implemented after the previous one was implemented, component-tested, and considered functional. A feature of the VxFS layout (actually only in the > v1 disk layout, but FreeVxFS doesn't support the historic VxFS v1 filesystems anyway) made this traditional approach impossible very early. The issue is that most of the metadata describing a filesystem is not directly stored in the superblock but in data blocks pointed to by regular inodes—this includes the inode table itself! (This is found by its containing extent to avoid endless recursion.)

To get past this point, a huge amount of code (almost half of the FreeVxFS implementation) had to be implemented at once, without previous testing of individual components. Of course this led to a number of very hard-to-debug problems and accounted for more than two-thirds of the development time.

7 Work in Progress and Future Plans

During the last month the FreeVxFS driver in the main kernel tree was steadily improved by bugfixes reported and/or fixed by the users. In addition support for different VxFS variants (block size, superblock locations) was added to support a broader range of target systems.

Ongoing major short-term development includes support for byte-swapping in the filesystem driver, allowing access to filesystems that were created on computers using a different byteorder than the accessing system (a feature VERITAS' driver is still lacking!) and a proper way to handle VxFS filesystems for HP-UX that have various small differences in the layout of important layout elements (e.g. the inode).

The most important long-term development project is to implement support for writing to VxFS filesystems. This feature is already functional in early stages, but fragments the filesystems so badly that it is of no practical use.

References

[AdmGuide] *VxFS System Administrator's Guide* VERITAS Inc. `http://ou800doc.caldera.com/ODM_FSadmin/CONTENTS.html`.

[Inode] *inode (vxfs) - format of a VxFS inode* Hewlett-Packard Company. `http://devresource.hp.com/STK/man/11.00/inode_vxfs_4.html`, (1997).

[Fs] *fs (vxfs) - fs format of VxFS file system volume* Hewlett-Packard Company. `http://devresource.hp.com/STK/man/11.00/fs_vxfs_4.html`, (1997).

Figure 1: Disassembly via `objdump`

```
0000000000075ba0 <vx_open>:
   75ba0:       8b 54 24 04             mov    0x4(%esp,1),%edx
   75ba4:       57                      push   %edi
   75ba5:       33 ff                   xor    %edi,%edi
   75ba7:       56                      push   %esi
   75ba8:       8b 02                   mov    (%edx),%eax
   75baa:       8b 70 28                mov    0x28(%eax),%esi
   75bad:       8b 40 24                mov    0x24(%eax),%eax
   75bb0:       83 f8 01                cmp    $0x1,%eax
   75bb3:       75 57                   jne    75c0c <vx_open+0x6c>
   75bb5:       8b 86 a4 01 00 00       mov    0x1a4(%esi),%eax
   75bbb:       25 00 f0 00 ff          and    $0xff00f000,%eax
   75bc0:       3d 00 90 00 00          cmp    $0x9000,%eax
   75bc5:       74 3d                   je     75c04 <vx_open+0x64>
   75bc7:       8b 44 24 10             mov    0x10(%esp,1),%eax
   75bcb:       a9 00 00 08 00          test   $0x80000,%eax
   75bd0:       75 2a                   jne    75bfc <vx_open+0x5c>
   75bd2:       6a 01                   push   $0x1
   75bd4:       56                      push   %esi
   75bd5:       e8 fc ff ff ff          call   75bd6 <vx_open+0x36>
   75bda:       83 c4 08                add    $0x8,%esp
   75bdd:       8b 46 5c                mov    0x5c(%esi),%eax
   75be0:       85 c0                   test   %eax,%eax
   75be2:       75 0a                   jne    75bee <vx_open+0x4e>
   75be4:       8b 46 58                mov    0x58(%esi),%eax
   75be7:       3d ff ff ff 7f          cmp    $0x7fffffff,%eax
   75bec:       76 05                   jbe    75bf3 <vx_open+0x53>
   75bee:       bf 4f 00 00 00          mov    $0x4f,%edi
   75bf3:       56                      push   %esi
   75bf4:       e8 fc ff ff ff          call   75bf5 <vx_open+0x55>
   75bf9:       83 c4 04                add    $0x4,%esp
   75bfc:       8b c7                   mov    %edi,%eax
   75bfe:       5e                      pop    %esi
   75bff:       5f                      pop    %edi
   75c00:       c3                      ret
   75c01:       83 c7 00                add    $0x0,%edi
   75c04:       bf 59 00 00 00          mov    $0x59,%edi
   75c09:       eb f1                   jmp    75bfc <vx_open+0x5c>
   75c0b:       90                      nop
   75c0c:       5e                      pop    %esi
   75c0d:       5f                      pop    %edi
   75c0e:       33 c0                   xor    %eax,%eax
   75c10:       c3                      ret
   75c11:       83 c7 00                add    $0x0,%edi
   75c14:       81 ff 00 00 00 00       cmp    $0x0,%edi
   75c1a:       81 ff 00 00 00 00       cmp    $0x0,%edi
```