Reprinted from the

Proceedings of the Linux Symposium

July 23rd–26th, 2008
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
Matthew Wilson, *rPath*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
Eugene Teo, *Red Hat, Inc.*
Kyle McMartin, *Red Hat, Inc.*
Jake Edge, *LWN.net*
Robyn Bergeron
Dave Boutcher, *IBM*
Mats Wichmann, *Intel*

Around the Linux File System World in 45 minutes

Steve French
IBM
Samba Team
sfrench@us.ibm.com

Abstract

What makes the Linux file system interface unique? What has improved over the past year in this important part of the kernel? Why are there more than 50 Linux File Systems? Why might you choose ext4 or XFS, NFS or CIFS, or OCFS2 or GFS2? The differences are not always obvious. This paper will describe the new features in the Linux VFS, how various Linux file systems differ in their use, and compare some of the key Linux file systems.

File systems are one of the largest and most active parts of the Linux kernel, but some key sections of the file system interface are little understood, and with more than than 50 Linux file systems the differences between them can be confusing even to developers.

1 Introduction: What is a File System?

Linux has a rich file system interface, and a surprising number of file system implementations. The Linux file system layer is one of the most interesting parts of the kernel and one of the most actively analyzed. So what is a file system? A file system "is a set of abstract data types that are implemented for the storage, hierarchical organization, manipulation, navigation, access, and retrieval of data." [4]

But a "file system" can also mean a piece of code, i.e., a Linux kernel module used to access files and directories. A file system provides access to this data for applications and system programs through consistent, standard interfaces exported by the VFS. It enables access to data that may be stored persistently on local media or on remote network servers/devices, or that may be transient (such as debug data or kernel status) stored temporarily in RAM or special devices.

The virtual file system interface and file systems together represent one of the larger (over 500 thousand lines of code), most active (averaging 10 changesets a day!), and most important kernel subsystems.

2 The Linux Virtual File System Layer

The heart of the Linux file system, and what makes Linux file systems unique, is the virtual file system layer, or VFS, which they connect to.

2.1 Virtual File System Structures and Relationships

The relationships between Linux file system components is described in various papers [3] and is important to understand when carefully comparing file system implementations.

2.2 Comparison with other Operating Systems

The Linux VFS is not the only common file system interface. Solaris, BSD, AIX, and other Unixes have similar interfaces, but Linux's has matured rapidly over the 2.6 development cycle. Windows, originally with an IFS model similar to that of OS2, has evolved its file system interface differently than Linux, and has a very rich set of file system features, but at the cost of additional complexity. A reasonably functional file system can be much smaller in Linux than in most other operating systems (look at `shmemfs` for example), including Windows.

2.3 What has changed in the VFS

During the past year, no new file systems were added, although one (`smbfs`) was deprecated. In the previous year, three new file systems were added: `ecryptfs` (allowing per-file encryption), `gfs2` (a new clustered

file system), and `ext4` (an improved, more scalable version of `ext3`). The file system area did improve dramatically though during the past year. From 2.6.21.1 to 2.6.25, the size of the VFS code grew almost 7% (from 38 KLOC to 41 KLOC). The total size of the file systems and VFS together (the `fs` directory and all subdirectories) grew about 6% (from 487 KLOC to 518 KLOC). 3612 changesets from almost 400 developers were added during the year (about 8.4% of the total kernel changesets), and resulting in adding or changing over 200,000 lines of kernel file system code over the year. There has been huge progress.

Interestingly, despite thousands of code changes, the VFS interface, the API needed to implement a fie system, changed only in minor ways, although file system drivers from the 2.6.21 source tree would require minor changes to compile on 2.6.25. The `exportfs` operations (needed to export a file system over the network via NFS) reworked its required methods (2.6.24). The `debugfs` and `sysfs` file systems also changed their external interfaces (2.6.24). The `debugfs` changes make it easier to export files containing hexadecimal numbers. Write-begin and Write-end methods were added to the VFS to remove some deadlock scenarios (2.6.24). New security operations were added to control mount and umount operations (2.6.25). SMBFS was deprecated (CIFS replaces it), but this did not affect the VFS interface. The kernel (2.6.22) now supports setting (not just getting) nanosecond inode timestamps via the new `utimensat(2)` system call. This call is an extension to `futimesat(2)` which provides the following:

- nanosecond resolution for the timestamps.
- selectively ignoring the `atime` or `mtime` values.
- selectively using the current time for either `atime` or `mtime`.
- changing the `atime` or `mtime` of a symlink itself along the lines of the BSD `lutimes(3)` functions.

A similar API call is being added to POSIX.

In the previous year, splice support was added. Splice is a mechanism to receive file data directly from a socket, and can dramatically improve performance of network applications like Samba server when they are reading file data directly from a socket. The name of a common structure element in the dentry changed as it moved into to the new `f_path` structure (2.6.20). The `readv` and `writev` methods were modified slightly and renamed to `aio_readv` and `aio_writev` (2.6.19). The inode structure shrunk (2.6.19), which should help memory utilization in some scenarios. There were changes to the `vfsmount` structure and `get_sb` (2.6.18). A new `inotify` kernel API (2.6.18) was added to fix some functional holes with the `DNOTIFY` ioctl. The `statfs` prototype was changed (2.6.18). Support for `MNT_SHRINKABLE` was added (2.6.17) to make implementation of global namespaces (such as NFS version 4 and CIFS DFS) possible. Shrinkable mounts are implicit mounts, and are cleaned up automatically when the parent is unmounted.

3 File Systems from a to xfs

There are many Linux file systems—each for a different purpose. Within the fs directory of the Linux kernel are 60 subdirectories, all but five contain distinct file system drivers: from adfs (which supports the Acorn Disk Filing System used on certain ARM devices) to XFS (a well regarded high performance journaling file system). Linux also can support out of kernel file systems through FUSE (the Filesystems in User Space driver).

4 File Systems

4.1 Types of File Systems

Conventionally we divide file systems into four types: local file systems, cluster file systems, network file systems, and special-purpose file systems. Local file systems are used to store data on a local desktop or server system, typically using the disk subsystem rather than network subsystem to write data to the local disk. Local file systems can implement POSIX file-API semantics more easily than network file systems, which have more exotic failure scenarios to deal with, and are limited by network file system protocol standards. Cluster file systems aim to achieve near-POSIX file-API semantics, while writing data to one or more storage nodes that are typically nearby, often in the same server room. Cluster file systems are sometimes needed for higher performance and capacity. In such file systems, which

often use SANs or network attached block storage, more disks can be connected to the system, and more actively used at one time, than could be achieved with a local file system running on a single system.

5 Local File Systems

5.1 EXT4

In 2.6.23 kernel, `ext4` added various scalability improvements including `fallocate()` support, increasing the number of uninitialized extents, and removing the limit on number of subdirectories. In addition, support for nanosecond inode timestamps was added (needed by Samba and some network services). The development of advanced snapshot and reliability features in ZFS have led to consideration of longer-term file system alternatives to `ext4`. One promising candidate is `btrfs` which was announced last year and is under development by Chris Mason at Oracle (although still experimental).

5.2 EXT2 and EXT3

With the obvious need to improve the scalability of the default local file system (`ext3` or `ext2` on many distributions), attention has focused on the follow-on to `ext3`, `ext4`. Despite this, there were 88 changesets added which affected `ext3` over the past year, from over 50 developers, changing over 1000 lines. This is a surprisingly high number of changes for a file system in "maintenance" mode.

5.3 JFS

IBM's JFS, which is in maintenance mode, had 45 changesets throughout the year (mostly global changes to structures, and minor code cleanup) but few new features.

5.4 XFS

In 2.6.23, XFS added a "concurrent multi-file data streams" feature to improve video performance and support for "lazy superblock counters" to improve performance of simultaneous transactions.

5.5 UDF

The "Universal Disk Format" is very important due to the explosion in numbers of writable optical drives. UDF added support for large files (larger than 1GB) in 2.6.22.

6 Network File Systems

6.1 NFS version 3

NFS version 3 defines 21 network file system operations (four more than NFS version 2) roughly corresponding to common VFS (Virtual File System) entry points that Unix-like operating systems require. NFS versions 2 and 3 were intended to be idempotent (stateless), and thus had difficulty preserving POSIX semantics. With the addition of a stateful lock daemon, an NFS version 3 client could achieve better application compatibility, but still can behave differently than a local file system.

6.2 NFS version 4

NFS version 4, borrowing ideas from other protocols including CIFS, added support for an open and close operation, became stateful, added support for a rich ACL model similar to NTFS/CIFS ACLs, and added support for safe caching and a wide variety of extended attributes (additional file metadata). It is possible for an NFS version 4 implementation to achieve better application compatibility than before without necessarily sacrificing performance. This has been an exciting year for NFS development with a draft of a new NFS RFC point release (NFS version 4.1) under active development with strong participation of the Linux community (including CITI and various commercial NFS vendors). The NFS version 4.1 specification is already 593 pages long. NFS version 4.1 includes various new features that will be challenging for the Linux file system to support fully, including directory oplocks (directory entry information caching), NFS over RDMA, and pNFS. pNFS allows improved performance by letting a server dispatch read and write requests for a file across a set of servers using either block- or object-based mechanisms (or even using the NFSv4 read and write mechanism). Some NFS version 4.1 features likely will be merged into the kernel by early next year, but their complexity has been challenging to the community.

6.3 NFS improvements

Over the past year, NFS has improved significantly, and has had more changes than any other file system. The SunRPC layer (which NFS uses for sending data over the network) now supports IPv6, although some smaller supporting patches are still being evaluated. Due to scarcity of IPv4 addresses in some countries, IPv6 support is becoming more important, especially as some government agencies are beginning to require IPv6 support in their infrastructure. NFS over RDMA, which provides performance advantages for workloads with large writes, is partially integrated into mainline (some of the server portions are not upstream yet). Server-side security negotiation is upstream. A new string-based mount options interface to the kernel has been added, which allows new options to be implemented in kernel without necessarily requiring `nfs-utils` (user-space tools) update, and eases long-term NFS packaging. Forced unmount support, which had been removed from NFS a few years earlier, was readded (2.6.23). This allows "umount -f" to better handle unresponsive servers. Also added to NFS in kernel version 2.6.23 was support for "nosharecache" which allows two mounts, with different mount options, from the same client to the same server export. When this mount option is not specified, the second mount gets the same superblock, and hence the same mount options as the first. With this new mount option, the user may specify different mount options on a second mount to the same export.

6.4 CIFS

IPv6 support was added (2.6.22). Additional POSIX extensions were added (2.6.22) to improve POSIX application compatibility on mounts to Samba servers. The most exciting changes to CIFS over the past year, though, have been the addition of Kerberos support and the addition of DFS support. MS-DFS is a mechanism for traversing and managing a global name space for files and is commonly used in larger networks. The Samba server already supported DFS, but the Linux kernel did not, until this year. Released earlier this spring, large amounts of interoperability documentation by Microsoft may allow us to improve our support more quickly, not just for newer servers, but also for older servers. Older dialects of SMB, sometimes more than 15 years old, are still in use in some places. This documentation is also making development of an in kernel SMB2 client implementation easier. Currently `smb2` support is being prototyped as a distinct module from `cifs`, to make it easier to make rapid changes, and because SMB2 is turning out to be much different than CIFS. Although sharing some information levels with SMB and CIFS, SMB2 has a much simplified set of commands that are largely handle- (rather than path-) based, and are even more efficient to parse, which should allow improved performance in the long run. SMB2 also allows improved asynchronous operation and request dispatching, while also adding better reconnection support via a new durable handle feature of the protocol. The prototype SMB2 client should be available (in experimental form) before the end of the year. Although the server for CIFS, Samba, is not in-kernel, it should be mentioned that with the recent release of the enormous amount of network interoperability documentation by Microsoft, the Samba team already has made great progress with the Samba 4 SMB2 server, already passing most functional tests.

6.5 AFS

There are multiple versions of AFS, the OpenAFS implementation which is more complete in function, but not in mainline kernel, and what a year ago was only a minimal implementation in-kernel. The AFS version in the mainline kernel has improved dramatically over the past year. In 2.6.22, support for basic file write was added, and support for directory updates (`mkdir`, `rename`, `unlink`, etc.) and `krb4` ticket support was added via RPC over AF_RXRPC sockets.

7 Cluster File Systems

There are two cluster file systems in the mainline kernel, GFS2 from Red Hat/Sistina and OCFS2 from Oracle. There are also two popular cluster file systems that are not in the mainline, but that are mentioned because they are often used in high-end compute clusters: Lustre (now owned by Sun) and IBM's GPFS. GFS2 supports more file system entry points than OCFS2 (which has worse locking and ACL support), but OCFS2 does support sufficient features to address the needs of some clustered databases.

7.1 OCFS2

In 2.6.22 OCFS2 added support for sparse files. Over the year the OCFS2 team and other kernel developers

added 247 changesets to OCFS2, more than 16,000 lines of code.

7.2 GFS2

This area had a busy year, with 243 changesets added: many small stability patches and bug fixes, but multiple performance enhancements were added as well.

8 Future Work

Going forward there are many challenges to address in the Linux File System area, but among the most interesting are the following: clustering improvements, new hierarchical storage management features, improved error handling and detection in the local file system, support for new network file systems (SMB2), and improved network file systems (NFS version 4.1).

8.1 Clustering

With the need for reliable server failover, and the need for dynamic reconfiguration of complex networks, clustering is becoming more important, but there is no clear winner among the cluster file systems, and two of the most popular choices still remain out-of-kernel. In addition, the NFS version 4.1 protocol adds the ability to support parallel NFS, to better distribute load across a set of NFS servers without requiring cluster file system software to be installed on all clients. Similar features are being investigated for CIFS. Samba server support for clustering is greatly improved through the work of Tridge, and others on the SOFS team, on `ctdb`. Ctdb has proven to be a very useful library for high performance cluster state management and recovery. NFS server support for running over a clustered file system also has improved in the past year through work by CITI and by IBM and others.

8.2 New Local File Systems

Among the biggest long term challenges in the file system area remains error handling and recovery. As disks get ever larger and yet error rates stay constant, error detection and recovery is reaching a critical point. One of the design goals of `btrfs` is to address this problem, as well as to improve HSM features. Many new systems now contain a mix of storage which includes disk and solid state. Since these devices perform very differently, local file systems must add features to optimize performance on hybrid systems which contain both. Whether in the long term we will need two local file systems, `ext4` and `btrfs`, due to performance differences on particular popular workloads, or whether one Linux file system will win and be used for most workloads remains to be seen. EXT4 scalability is improved but would require substantial changes to handle ever increasing disk errors on ever larger disks as well as `btrfs` already does. In some operating system, the file system and volume manager layers of the operating system are more tightly coupled than in Linux. This eases the addition of better support for dynamic reconfiguration of disk subsystems when failing disks are added or removed on the fly. The development of `btrfs` may open up discussion of changes to the volume management layer as well as changes to Linux to support DMAPI, the standard for disk management used by many storage management applications. Adding support for part or all of DMAPI to the VFS and `btrfs` or `ext4` would allow for improved backup and disk management. Currently a subset of DMAPI is supported but on XFS only.

8.3 New Network File Systems

An SMB2 file system prototype, written by the author, is being coded and tested currently. Since SMB2 is significantly different than SMB and CIFS protocols in the way it handles path names, and in the way it handles file handles, less code could be shared between it and the existing cifs module, so it is being written as a distinct module. This also allows it to be updated quickly without impacting the existing cifs module. SMB2 will become more important in the coming year since it is the default network file system for current Microsoft servers and clients, and matches reasonably well to Linux. NFS version 4.1 also includes new features which will need to be explored in the Linux VFS, including how to support directory oplocks (directory delegations).

9 Legal Statement

References

[1] O. Kirch. Why NFS Sucks. *Proceedings of the 2006 Ottawa Linux Symposium*, Ottawa, Canada, July, 2006. `http://ols.108.redhat.com/reprints/kirch-reprint.pdf`

[2] Linux CIFS Client and Documentation. `http://linux-cifs.samba.org`

[3] S. French. Linux File Systems in 45 minutes: A Step by Step Introduction to Writing or Understanding a Linux File System. `http://pserver.samba.org/samba/ftp/cifs-cvs/ols2007-fs-tutorial-smf.pdf`

[4] File System. Wikipedia, The Free Encyclopedia, Retrieved 28 April 2008. `http://en.wikipedia.org/wiki/Filesystem`

[5] Linux Weekly News, API changes in the 2.6 kernel series `http://lwn.net/Articles/2.6-kernel-api/`

[6] Kernel Newbies. Linux Changes report. `http://kernelnewbies.org/LinuxChanges`