Staying in sync with a network filesystem

CONNECTIONS

Tired of copying and recopying files from your laptop to the office file server? Maybe you need an automated offline filesystem, such as OFS.

BY CARSTEN KOLASSA, FRANK GSELLMANN, TOBIAS JÄHNEL, PETER TROMMLER

U sers routinely copy documents to their laptops, edit the files from the road, and save the changes centrally when they get back to the office.

Unfortunately, it is all too easy to lose files, overwrite changes, or forget which version is the most recent. Windows provides an offline file storage option to address this problem, and several alternative tools are also available (see the "Similar Approaches" box).

Now a new project brings the offline storage option to Linux: OFS, the offline filesystem [1]. (Because it began at the Georg-Simon-Ohm University in Nuremberg, Germany, OFS also stands for Ohm Filesystem.)

OFS

To start, simply select the directories you need, and then the offline filesystem copies the contents of these directories to a cache on your local disk. Even if you don't have a connection to the server, you will still appear to be working on the network filesystem. In reality, you will be working with the copies in the cache. Paths stay the same whether or not you have a connection to the network.

When the connection becomes available again, the offline filesystem automatically launches a reintegration session to write the changes out to the server.

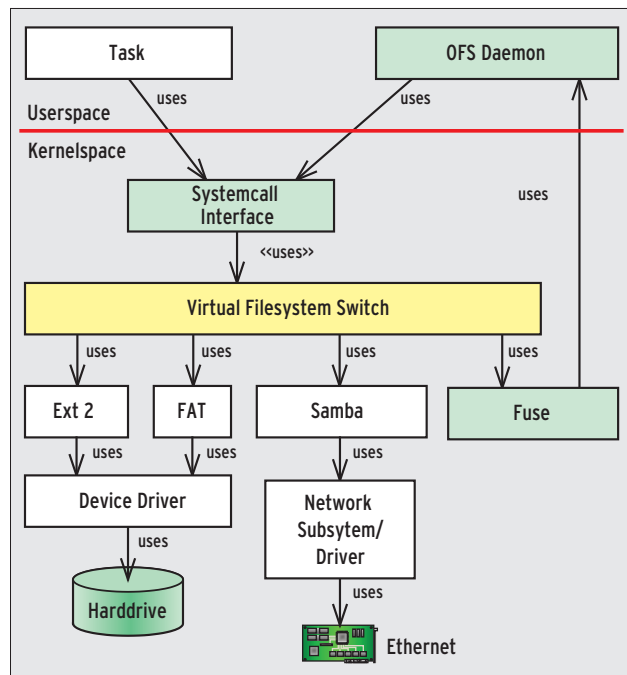OFS is not a network filesystem in the true sense of the word. In fact, it is a

Figure 1: The OFS offline filesystem relies on FUSE for kernel interaction.

layer between the network filesystems (for example, NFS or Samba) and a user view, which means that you can combine OFS with any filesystem.

How It Works

OFS code runs completely in userspace and relies on FUSE (Filesystem in Userspace [2]). FUSE provides an interface the kernel uses to forward file access to userspace programs.

The FUSE interface mainly consists of a kernel module, *fuse.ko*, and the Libfuse library, both of which are included with any recent Linux distribution. The kernel's VFS (Virtual Filesystem Switch) accepts the FUSE kernel module as a filesystem. FUSE forwards calls to the OFS daemon in userspace via the */dev/fuse* device file.

To receive data, OFS relies on Libfuse and the C++ bindings by the Fusexx project [3] (Figure 1). Each action that affects a file or directory on an OFS filesystem triggers a function call though FUSE to the OFS daemon.

In addition to the OFS daemon shown in Figure 1, the OFS project also provides a mount helper and a file browser plugin (Figure 2). To communicate, the file browser plugin, which acts as a user interface to the OFS daemon, uses D-Bus [4]. This design makes it fairly simple to extend OFS by adding more file browser plugins, no matter which desktop you need to support.

The mount helper, *mount.ofs*, mounts the remote filesystem, for example, a Samba share, and launches the OFS daemon. The OFS daemon resides in user-
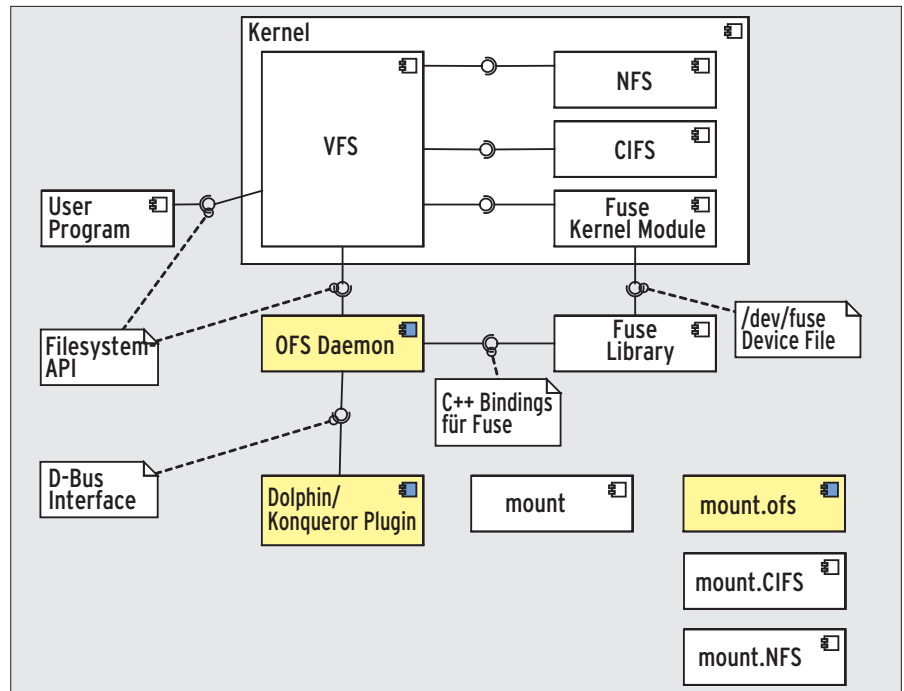
Figure 2: An overview of the OFS environment. The Dolphin and Konqueror plugins use D-Bus to communicate with the OFS daemon, which uses both the filesystem API and FUSE to talk to the kernel.

space and accesses the remote filesystem. As with any normal application, OFS uses the standard filesystem API for this access.

To allow this to happen, the mount helper does not mount the remote filesystem at the location the admin specified in the call, but under */var/ofs/-remote/URL Hash*.

The original mount point is controlled by FUSE. Thus, OFS is completely independent of the remote filesystem and will work with any implementation supported by the Linux kernel.

The OFS daemon's internal structure is service oriented, making it easily extensible. In the simplest case, the daemon passes all requests on to the remote filesystem itself. However, it is also responsible for caching and maintaining all persistent data.

When a user enables caching – for the whole filesystem, or for individual directories – OFS creates a copy of any file that is opened in the cache. When a user works with the file, the OFS daemon writes the changes, both to the cache copy and to the remote file on the server

Similar Approaches

Windows Offline Files: The motivation for the OFS project was the desire to create a Linux counterpart to Windows offline files. Microsoft introduced the feature to its operating systems in Windows 2000, and it is integrated with the Synchronization Center in Vista. Windows offline files makes files stored on an SMB share available offline. Users can access the share even if their computer doesn't have a connection to the server. Once the share is available again, Windows will synchronize automatically or at the user's request [5].

Coda: As early as the 1980s, a team at Carnegie Mellon University (CMU) started to create a network filesystem designed to make lost network connections invisible to

users. Coda, which was intended as the successor to the Andrew File System (AFS), can bridge short-term network failures, letting users work without a network connection. The Coda programmers developed some excellent ideas, although the filesystem is not totally mature [6].

Intermezzo: Intermezzo, which is also maintained by CMU, is similar Coda but with a far simpler design. Whereas Coda only accesses the cache if it does not have a connection, Intermezzo always uses the cache. The filesystem synchronizes the client cache with the server content by periodically polling the server or following a server request. The Intermezzo filesystem is no longer under active development; in

fact, it was dropped from Linux kernel version 2.6, and it is not suitable for production use in most cases.

Version management: Offline filesystems cannot replace a full version management system like Subversion [7]. Version management systems provide a more extensive range of revision control features, but they are more complex and not as convenient as simple offline storage tools.

Synchronization Tools: Unison [8] and other products synchronize the contents of local and remote directories; however, users have to handle synchronizing their copies with the server content, even if they have an active connection.
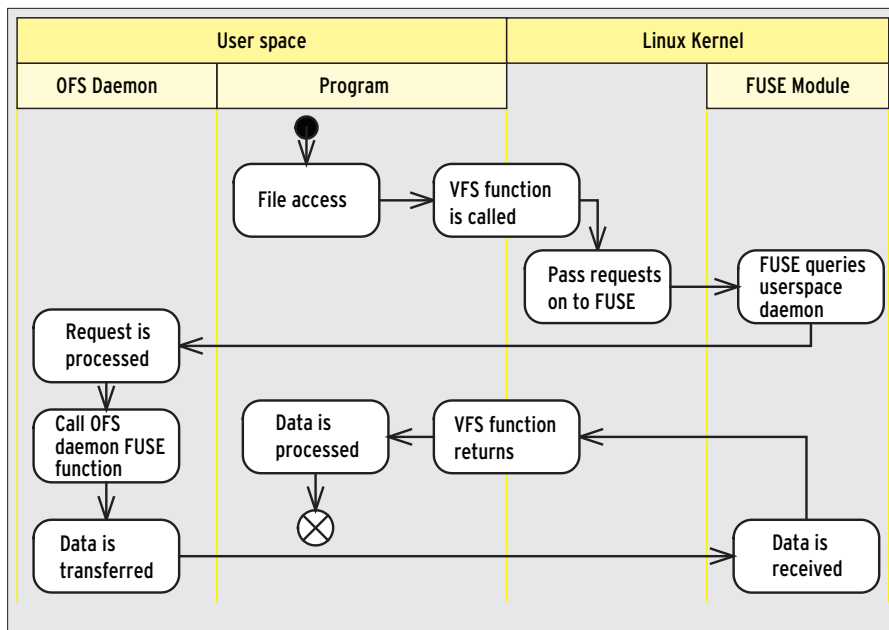
Figure 3: When a program accesses a file, the kernel-side VFS detects that FUSE is responsible. The FUSE module passes the call to the OFS daemon and receives its response.

if it is available. See Figure 3 for the individual steps for opening, reading, and writing a file.

Practical Applications

To create an OFS-managed mount point, or – to put this a different way – to mount a share via OFS, use the following command:

```
mount -t ofs type://server/share ↵
/mountpoint
```

The command

```
mount -t ofs ↵
smb://fileserver.comp/data ↵
/network/data
```

mounts the SMB share *data* on the server *fileserver.comp* in the local */network/data* directory. The *mount.ofs* mount helper uses FUSE to let the OFS daemon intercept calls to */network/data*. At the same time, the mount helper breaks down the URL and executes an additional mount command to mount the remote directory:

```
mount -t smb ↵
//fileserver.comp/data ↵
/var/ofs/remote/URL_hash
```

Each OFS-managed mount point includes a state indicating whether or not the share is available and directing access to the cache or the share itself. When the network cable is removed, Linux uses D-Bus to trigger an event, which OFS then queries before setting the filesystem status to *offline*.

In the future, the developers would like to integrate more features, such as a timer to identify server problems or a polling component to detect the current online state.

Think Sync

OFS uses synchronization in two situations: to keep the cache up to date while the server connection is up and to reintegrate any local changes with the server when a lost connection is reestablished.

Alternative tools, such as Coda and Intermezzo, provide software on the server side to let the server report changes to all relevant clients. OFS, of the other hand, is a client-only solution – it is the user's responsibility to trigger the cache update process manually.

When updating the cache, OFS relies on the files' modification timestamps. Then the OFS daemon compares the timestamp of the original file in the share to the copy in the cache. If the remote file has changed, OFS copies it to the local cache.

To synchronize, OFS must perform a full search of the offline directory tree. Because this process stresses the network, the hard disk, and the CPU, the developers decided to do without full automation based on polling.

When a program issues an *open()* system call, the update process is triggered. At that point, OFS checks to see whether the file in its offline directory is up to date; if not, it downloads the latest version from the server. OFS uses the local cache to answer read access requests, whereas write requests modify both the cache and the share (Figure 4). This approach guarantees both fast read access and up-to-date files.

When the server connection is down, OFS logs all write accesses handled by the cache. Each entry in the logfile contains both the path to the file and a parameter specifying the change. The possible values are: created, deleted, and modified.

Once the server connection is restored, OFS works its way through the logfile,
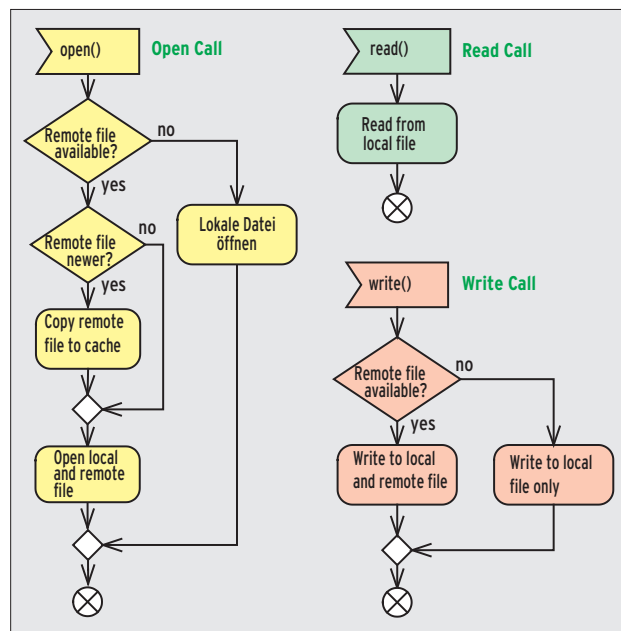
Figure 4: When a file is opened (left), OFS updates the copy in the local cache if possible. Read requests (top right) are served directly from the cache to improve performance. Write requests modify both the cache and (if possible) the original on the server (bottom right).

performing each of the steps on the server side: creating new files, deleting existing files, or uploading the cached version to the server.

During reintegration, a file the user has modified locally might have been changed on the server, too. To avoid simply overwriting the file, OFS first checks how current the remote file is. If the modification time is later than the last synchronization time, the remote file must have been changed in the meantime. In this case, OFS launches its conflict-resolution tool.

In some cases – for example, if one user has modified the local file while an-

other user has simply renamed the version on the server – the conflict is easily resolved. However, some conflicts are impossible to resolve automatically, such as cases in which two users modify the file in different ways while it is offline. In this situation, OFS leaves the decision to the users. A diff mechanism analyzes both versions of the file, and a preview shows the results of the proposed conflict resolution. (The OFS configuration decides which diff tool to use.) Text files are typically fairly easy to merge, with the exception of a situation in which both sides have changed the same part of the file. Because it is more difficult to

display and merge differences in binary files, the only approach is to choose one of the versions.

The *KFilePlugin* plugin for Dolphin and Konqueror adds an *OFS* tab to the *Properties* dialog field, which is displayed when you right-click a file or directory and select *Properties*. By selecting the OFS tab, users can add a directory to the local cache or write changes out to the server. This GUI plugin is not a required part of OFS. Machines without a GUI can still use OFS at the command line.

Future

OFS is not a mature product, although the current version is usable if you accept its limitations. The roadmap features a number of extensions. The modular design supports the introduction of more agent classes, for example, to provide more precise details on the availability of a remote filesystem. ∎

Caching

In OFS, the local cache mirrors the share. When the user chooses which directories to access while on the road, OFS creates a full mirror copy of only the required components. Other offline filesystems either copy the whole share or select the files based on some form of babysitting algorithm, for example, the most frequently changed or the most recently used files.

Because the OFS daemon always creates the whole path from the root of the share to the directory for which the user needs offline access, OFS can redirect access to the cache directly in case of connection loss, without the need to emulate missing parent directories.

Internally, OFS supports three directory trees: a local cache (*Cache-Dir*), a remote share directory (*Remote-Dir*, where OFS mounts the share), and the user's working directory (*Working-Dir*). The complete cache and the mount point for the remote share are local directories created by OFS itself.

Entries in the OFS daemon's configuration file decide where OFS creates the directories; the default location is */var/ofs*. The directory name is a hash of the remote share's URL. This approach avoids conflicts and ensures that the admin can easily copy or move the internal directories.

Attributes

For the most part, OFS interacts with applications just like any other filesystem. Its special features include a flag that specifies whether a file should be available offline if the connection is lost. Many filesystems, AFS for example, have their own tools for setting these parameters. OFS, however, removes the need for special tools by relying on extended file attributes, which the user can set and read through standard Linux file utilities.

Extended file attributes make it possible to add metadata to files (as long as the filesystem supports this). The kernel does not handle the *get* and *set* commands itself but passes them directly to the filesystem. It is thus the filesystem's responsibility to process or save the attributes. FUSE supports extended file attributes and offers callback functions for setting, deleting, reading, and listing attributes.

OFS relies on this feature to implement its own custom attributes, which serve only to exchange information between the file-

system and the shell. OFS does not save them and does not pass them on to the underlying filesystem; you might say they are virtual attributes. The current version of OFS supports two attributes in the *ofs* namespace:

- *ofs.offline* indicates whether a file is available offline. Setting this attribute triggers the update process and adds the tree to the list of directories available offline.

- *ofs.available* indicates whether the file is available right now, or whether the cached version is in use. In other words, it tells you whether the server connection is available. This attribute is read-only.

In both cases, the only two states are "set" (yes) and "not set" (no). *setfattr -n attribute_path* sets the state, and *getfattr -n attribute_path* reads it. To delete the attribute completely, you can use *setfattr -x attribute_path*.

INFO

[1] OFS (Offline filesystem): *http://offlinefs.sourceforge.net*

[2] FUSE: *http://fuse.sourceforge.net*

[3] Fusexx, C++–Bindings for FUSE: *http://portal.itauth.com/2007/07/07/c-fuse-binding*

[4] D-Bus: *http://dbus.freedesktop.org*

[5] Windows offline files : *http://support.microsoft.com/kb/307853/*

[6] Coda: *http://www.coda.cs.cmu.edu*

[7] Subversion: *http://subversion.tigris.org*

[8] Unison: *http://www.cis.upenn.edu/~bcpierce/unison/*

THE AUTHOR

Peter Trommler is Professor of Theoretical Computer Science, Computer Communications, and Information Security at the Georg-Simon-Ohm University in Nuremberg, Germany. He has been working on Unix for 18 years and on Linux for 10 years.

Carsten Kolassa has worked in the Linux community for many years, focusing on embedded topics, networks, and security.

Frank Gsellmann is currently developing the resyncronisation mechanism of OFS project as his Master's thesis. Tobias Jähnel laid the foundations for the project for his thesis; he now works for EB Automotive: *www.elektrobit.com*