

RedirFS

Frantisek Hrbata
<frantisek.hrbata@redirfs.org>

www.redirfs.org

October 22, 2007

Abstract

The RedirFS or redirecting file system is a new layer between virtual file system switch (VFS) and file system drivers. It is implemented as an out-of-kernel module for Linux 2.6 and it provides framework allowing modification of file system calls in the VFS layer. The RedirFS by itself does not provide any additional functionality and if it is loaded into the Linux kernel, it just occupies some memory space and it does practically nothing. Now you maybe ask yourself a question: "So what is it good for then?". The RedirFS is intended to be used by so-called filters. Filter is a linux kernel module (LKM) that uses the RedirFS framework. Each filter can add some useful functionality to the existing file systems like transparent compression, transparent encryption, merging contents of several directories into one, allowing writing to a read-only media and others. Filter can set pre and post callback functions for selected file system calls (e.g. read or open) and include or exclude directories or single files for which its callback functions will be called. Every filter has its priority - a unique number that defines in which order will the filters be called. The RedirFS manages all filters and calls their callback functions in order specified by their priorities. While the RedirFS is located in the VFS layer, filters can be used generally in all file systems (e.g. ext2, nfs, proc). In addition, the RedirFS allows filters to be on-the-fly registered and unregistered.

- provides framework allowing to redirect file system calls in the VFS layer
- is implemented as an out-of-kernel module for Linux 2.6
- is able to register, unregister and manage one or more filters
- allows filter to set its callback functions on-the-fly
- allows filter to include and exclude its paths on-the-fly
- allows filter to forward data from pre to post callback function
- allows filter to attach its private data to VFS objects
- allows filter to do subcalls for selected file system calls
- calls pre and post callback function of filters in fixed order specified by their priorities (call chain)
- reacts on return value from filter and it is able to interrupt filters call chain
- redirects only operations selected by one or more filters, all other operations go directly to the file system with no time overhead
- modifies only VFS objects which belong to paths selected by one or more filters

1 RedirFS Objects

The RedirFS is based on replacing operations of the VFS objects - file, dentry, inode and address_space. The RedirFS creates for each file, dentry and inode object a corresponding RedirFS object. To avoid confusion with the VFS objects, RedirFS objects are called rfile, rdentry and rinode. The RedirFS object exists along with the corresponding VFS object and it is connected with the VFS object via the VFS object operations. Each of the RedirFS object contains new operations for the VFS object and all other information which are needed to call all interested filters. The RedirFS objects, besides that they are connected with VFS objects, are also connected with each other. The rfile objects contains pointer to the rdentry object. The rdentry object contains pointer to the rinode object and a list of all rfile objects opened for it. This list is used when the operations of VFS objects are restored to the original operations so the RedirFS can easily find out which VFS files were modified. And the rinode object contains list of all rdentries created for it. To synchronize the RedirFS objects creation, the rdentry object is created using the dentry d_lock and rinode is created using the inode i_lock. The rfile object creation is unique and it cannot be created twice and therefore no synchronization is needed.

2 VFS Object Operations Replacement

The RedirFS creates new operations for each VFS object. This allows RedirFS to set different operations for different VFS objects. For example disk based file systems have only one set of operations for regular file inodes. Each regular file inode for such file system has pointer to the same inode operations in the file system driver. RedirFS creates for each such inode a new set of inode operations.

The VFS layer does not know anything about the RedirFS. It just calls operations which are set for its objects. As mentioned previously, the RedirFS replaces those operations so that VFS layer can call functions in the RedirFS framework. For example when a filter is interested in some inode operation, the RedirFS creates for this inode a rinode object. This rinode object contains new inode_operations and a pointer to the original inode_operations. The new inode_operations in the rinode object are initialized with values of the original inode_operations, operations in which filter is interested are replaced with RedirFS operations. The new inode_operations within the rinode object are assigned to the inode object and a pointer to the old inode_operations is stored in the rinode. The new operations stay the same as the original operations until one or more filters request to set pre or post callback function for the specific operation. Thanks to this approach there is no time overhead for operations for which filters did not set pre or post callbacks. However there are several operations which need to be replaced every time. This is because the RedirFS needs to keep track of created and deleted VFS objects so it can replace operations for newly created objects and on the other hand restore operations when the VFS objects are deleted. Below is a list of operations that need to be redirected to the RedirFS each time. Note: all – all file types, dir – only directory.

- file_operations: open, release – all, readdir – dir
- dentry_operations: d_iput, d_release – all

- `inode_operations`: `mkdir`, `create`, `link`, `symlink`, `mknod` – `dir`, `lookup` – `all`

3 RedirFS Connection with VFS

While VFS object pointer to its operations set points to the operations stored within the RedirFS object, the RedirFS can easily obtain the RedirFS object corresponding to the VFS object through the `content_of` kernel macro. This means that VFS and RedirFS objects are connected through operations. The RedirFS object is deleted when filters are no more interested in the VFS object operations. This can lead to a situation when the VFS layer calls the RedirFS operation but right before the RedirFS object is obtained, it is deleted. In this situation RedirFS knows that the VFS object operations were set back to the original operations and it just calls the original operation. If the original operation is not implemented by the file system, the RedirFS calls proper default VFS operation or returns an error.

How RedirFS finds out that the operations were set back to the original ones? The trick is, that RedirFS sets for each new set of operations one fixed RedirFS operation. For `inode` it is `lookup`, for `dentry` `d_lput` and for file open operation. So the RedirFS just finds out whether this fixed operation is set in the VFS object operations set or not. If it is, it is safe to use `content_of` macro and get the corresponding RedirFS object. In addition, RedirFS is using `rcu` for synchronization when pointers to the VFS objects operations are changed. This ensures that nobody else will manipulate with the pointer while RedirFS is comparing pointers. Moreover, RedirFS objects are using proper reference counting. Thanks to these approaches, VFS object operations pointer can be safely changed and RedirFS objects can be created and deleted on-the-fly. The RedirFS is using the fact that pointer assignment in Linux kernel is atomic.

4 Walking Through the Dentry Cache

The RedirFS is replacing operations of VFS objects for paths selected by filters. This is done by walking through the VFS `dcache`. Since `dcache_lock` is exported by the Linux kernel, the RedirFS can safely go through the selected `dentry` subtree and replace operations for `dentry` and `inode` objects while each `dentry` object has a pointer to its `inode` object. There is one exception for so-called negative `dentries`. Negative `dentry` does not have a pointer to the `inode` and it is used to speed-up path lookup for files which do not exist. There is no safe way how to find out which file objects are already opened for `dentry` object. This means that RedirFS is not able to modify operations of files that were opened before `inode` operations for selected file were replaced. So filters can set their pre and post callback function only for files opened after the `inode` operations for selected files were replaced.

For going through the `dcache`, the RedirFS implements the general `rfs_walk_dcache` function. This function goes through `dentry` subtree, starting by `dentry` object passed to it as an argument and for each `dentry` in this subtree it calls callback functions passed to it as arguments. This function uses for each level in the subtree the parent `inode` `i_mutex` to make sure that nobody else can add or remove objects from this subtree level while the RedirFS is modifying objects operations. The RedirFS uses this function for replacing,

restoring and setting VFS objects operations.

During walking through the dentry cache the dentry operations are replaced for each dentry and inode operations along with address space operations for each inode. The address space operations are replaced only for regular files because they have their address space operations set by a file system and stored in the `i_data` inode attribute. For other file types (like block device which changes the whole `address_space` object in the file open function) it is now not possible to replace their address space operations. Besides that, the RedirFS has to use the `truncate_inode_page` function to invalidate the page cache for inode every time the operations are changed. File operations in the inode objects are replaced with generic RedirFS file operations which implement only open operation. File operations for file are replaced after the VFS file object is created because file operations can be changed during open call. This is used by the special files like char device or fifo. RedirFS is using the same principle for replacing file operations like special files do.

5 Filters Call Chain

Each RedirFS object contains pointer to the so-called filters call chain. Filters call chain is a list of filters which are interested in one or more operations of the VFS object sorted according to their priorities. This list tells RedirFS which Filters have to be called before (pre callbacks) and after (post-callbacks) the original operation. Each filter can finish or stop the proceeding operation in its pre callback function. This means, that the original operation will not be called. Filter can finish proceeding operation by returning an error code or by finishing it. Operations can be finished by for example RedirFS subcalls. Filters call chain for newly created RedirFS object is inherited from his parent. This happens when new VFS object is created (e.g. new file is created).

6 RedirFS Subcalls

Subcalls are RedirFS functions for selected VFS operations. Subcall calls only filters which are in the filter call chain past the filter which called the subcall. This allows filter to call the same VFS operation with its own or different arguments.

7 Private Data

The post callback functions will be called for each filter for which its pre callback function was called. This is because filter can attach its private data per operation in the pre callback function and it has to detach them in the post callback function. Filters are also allowed to attach their private data to each VFS object. When the VFS object is going to be deleted, the RedirFS will notify each filter which has its private data attached to this object to detach them. Filter attaching its private data to the VFS object via RedirFS has to also provide along with its data the callback function which will be called when the VFS object or corresponding RedirFS object will be deleted. Private data for filters are kept in a list in the RedirFS object corresponding to the VFS object.

8 Path Management

Paths selected by filters are in the RedirFS represented by the rpath structure. Filters can include or exclude directory subtrees or even single path (file or directory). The rpath objects are connected in trees and the root of each tree is stored in the global path_list list. When a filter wants to add a new path for its callbacks, RedirFS checks if corresponding rpath already exists (possibly added by some other filter). If it does not exist, the RedirFS creates new rpath object, adds it to the rpath tree and if this new rpath has a parent rpath, it inherits all filters from the parent path.

As mentioned previously, each RedirFS object has pointer to the filters call chain. Filters call chain represents list of filters attached to the rpath. Each RedirFS object has pointer to the filters call chain of rpath to which it belongs. The RedirFS implements general rfs_path_walk function which goes through all rpaths in the rpaths tree and for each rpath it calls a callback function. Starting rpath and the callback function are passed as arguments to this function. If starting rpath is NULL it goes through all rpaths in all trees. Each rpath object contains full pathname of the path it represents. This is used for finding rpath object for the corresponding path entered by a filter in the rpath tree. The rpath object contains pointer to the dentry object to which it belongs. The dentry object is found with the path_lookup function exported by the VFS. So when RedirFS has the rpath object, it knows the root dentry object which is then used in the rfs_walk_dcache function to replace, restore or set VFS objects operations. The path_list is protected by the path_list_mutex which means that only one process can manipulate rpath tree, use the rfs_walk_path and rfs_walk_dcache function. To ensure that RedirFS will change operations that belongs only to the selected rpath, each rdentry object contains rd_root flags. If this flag is set, it means that this dentry object belongs to the other rpath and rfs_walk_dcache has to skip subtree with this dentry.

While filter can select single path, the rpath object has to be able to handle a situation when one filter includes the path subtree and other filter includes the same path but as a single path. In this case rpath has two filters call chains. One local chain with both filters only for objects that belongs to the single path and second chain with one filter for all objects in the subtree. When rpath has the same filters in the filters call chain as its parent, it is deleted and all RedirFS objects switch to the parent rpath. If rpath has no filters in the filters call chain, it is deleted together with all RedirFS objects which belong to this rpath and operations of all VFS objects for this rpath are restored to the original operations. RedirFS objects are moving from one rpath to the other as the rpaths are created and deleted.

The rpath management is unfortunately more complicated because filters can also exclude paths subtrees and single paths. This means that rpath object has four filters call chains - global include, global exclude, local include and local exclude. Before rpath is removed, RedirFS also has to check the exclude chains.

9 Sysfs Interface

The RedirFS creates the same basic attributes for each filter in the sysfs file system. For each filter is created a directory /sys/fs/redirfs/<filter name>. This directory contains the following files (attributes):

- active – activated or deactivated filter
- paths – list of included and excluded paths, path can be set via this file
- control – flags if filter is accepting setting via the sysfs interface
- priority – filter’s priority

10 Filters

Filter in the RedirFS is represented by the filter structure. Each filter has a name, a unique priority number, a set of pre and post callback operations and a set of paths. Filter’s name is used for the directory name in the sysfs file system. Filter can register a callback function to receive settings via the sysfs interface but it can ignore it and use its own way how to set the paths. Pre and post callback filter’s functions are stored in arrays - `f_pre_cbs` for pre callbacks and `f_post_cbs` for post callbacks. As mentioned previously, each path has a filters call chain. RedirFS makes a new operations for VFS objects for selected rpath by unification of all callback functions of all filters in the filters call chain and it goes through the dentry cache and set these new operations to VFS objects.

References

- [1] Bovet, P. B., Cesati, M.: Understanding the Linux Kernel. 3rd Edition, U.S.A., O’Reilly 2005
- [2] Love, R.: Linux Kernel Development. 2nd Edition, Indianapolis, Indiana, Novel Press 2005
- [3] Alessandro Rubini, Jonathan Corbet: Linux Device Drivers, 2nd Edition, U.S.A., O’Reilly 2001