ClamFS Internal

expire and LRU based cache and FUSE in ClamFS

by eqmcc

eqmcc@http://blog.csdn.net/eqmcc

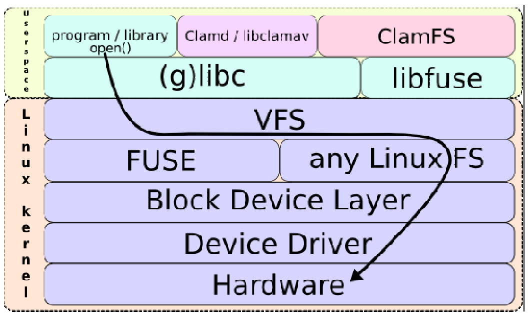
Description

this document will talk about on-access scan in clamav with the support from clamfs.

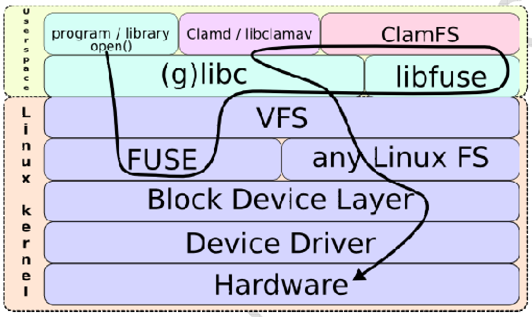
1. about clamfs

   ClamFS(http://clamfs.sf.net/) is a user space file system based on FUSE.

   compared to normal vfs file operation from glibc, clamfs go through extra layers in both user space and kernel space to hook to the clamav daemon asking for virus scan service.

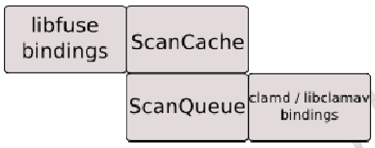


vfs file open

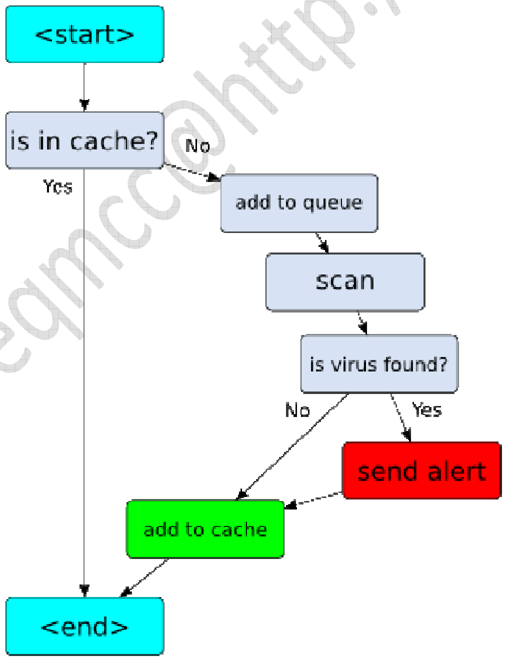


clamfs file open

in addition to above internals, clamfs also implements a internal cache (LRU with time-based and out-of-memory expiration) to speed up the file scan:



the whole scan flow is as bellow:



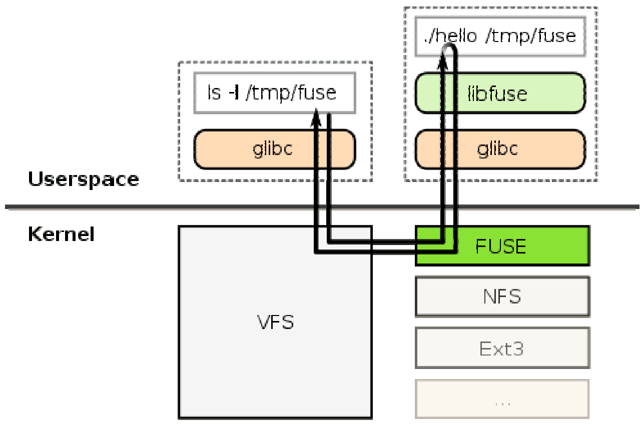
Note: all the diagrams are referred from the official web page of ClamFS.

eqmcc@http://blog.csdn.net

2. about FUSE

FUSE(**F**ilesystem in **Use**rspace, http://fuse.sourceforge.net/) can be employed to make it possible to implement a fully functional file system in a user space. FUSE was originally developed to support ***AVFS*** but it has since became a separate project. FUSE is used in ZFS， glusterfs, luster and clamfs, etc…

the path of a filesystem call (e.g. stat) in FUSE is as following:
(referred from http://zh.wikipedia.org/wiki/File:FUSE_structure.svg)



Data structures

TBD

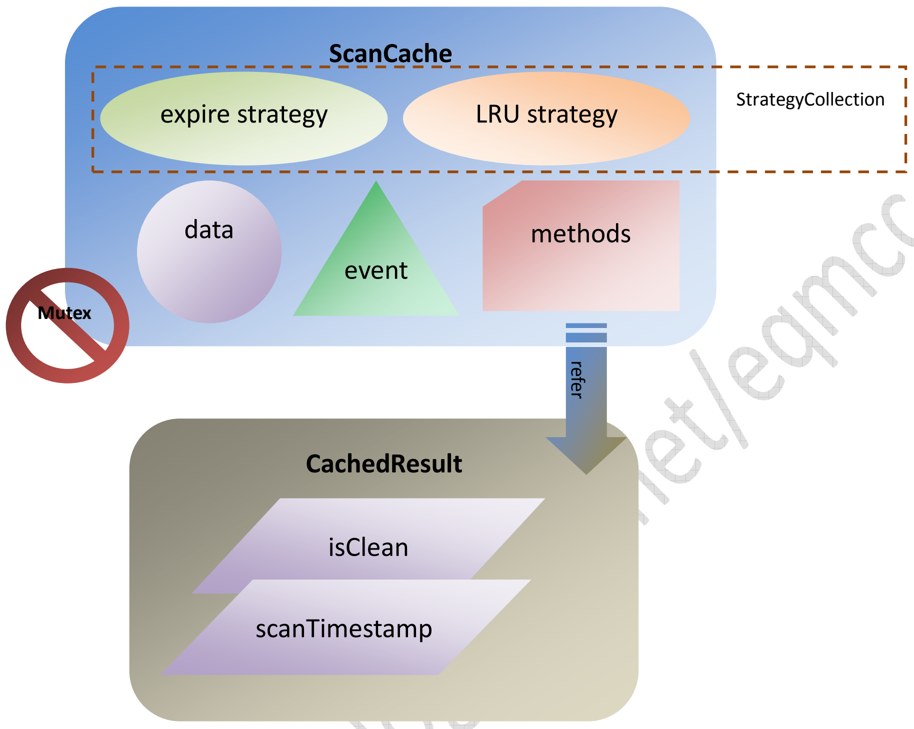
Test case

TBD

eqmcc@http://blog.csdn.net/eqmcc

Cache system in ClamFS

There is a cache system in clamfs that caches the scan results to improve the performance. The cache has two strategies and few other components as bellow:



eqmcc@http://blog.csdn.net/eqmcc

The two strategies will be triggered when there's an event(add, update, etc..) against the cache, for example, when an add method is invoked, the two strategies will be run to update the cache according to expire time and LRU check point.

Any file accessed in clamfs will be first checked in cache to seek the quick match before doing any further scan delegated to clamav.

Class construction

ScanCache is derived from ExpireLRUCache
ExpireLRUCache is derived from AbstractCache
AbstractCache defines basic elements of a cache:

- Event: FIFOEvent(Add/Update/Remove/Get/Clear)
- data : DataHolder _data
- StrategyCollection: TStrategy _strategy
- methods: add/update/remove/has/clear/size/forceReplace/getAllKeys
- mutex: TMutex _mutex

Also event's handling is delegated to each strategy collection's corresponding handling function

construction of Object

1. FIFOEvent is derived from AbstractEvent who is specialized by FIFOStrategy and AbstractDelegate

2. this FIFOStrategy specialized by AbstractDelegate is derived from DefaultStrategy which have FIFO behavior with following methods:

```
notify
add
remove
operator =
clear
empty
```

which will operate against the delegates

3. AbstractEvent using FIFOStrategy has following methods operating against FIFOStrategy:

```
operator +=
operator -=
operator ()
notify
notifyAsync
```

4. in AbstractCache::initialize()

the events of cache and the delegated methods have following maping:

Event	delegated method	instance method
Add	TStrategy::onAdd	StrategyCollection::onAdd
Update	TStrategy::onUpdate	StrategyCollection::onUpdate
Remove	TStrategy::onRemove	StrategyCollection::onRemove
Get	TStrategy::onGet	StrategyCollection::onGet
Clear	TStrategy::onClear	StrategyCollection::onClear
IsValid	TStrategy::onIsValid	StrategyCollection::onIsValid
Replace	TStrategy::onReplace	StrategyCollection::onReplace

5. adding strategies to object

```
this->_strategy.pushBack(new LRUStrategy<TKey, TValue>(cacheSize));
this->_strategy.pushBack(new ExpireStrategy<TKey, TValue>(expire));
```

two strategies are added to ExpireLRUCache object:

```
LRUStrategy
ExpireStrategy
```

the two strategies are derived from AbstractStrategy in which defined basic methods of any strategy:

```
onUpdate
onAdd
onRemove
onGet
onClear
onIsValid
onReplace
```

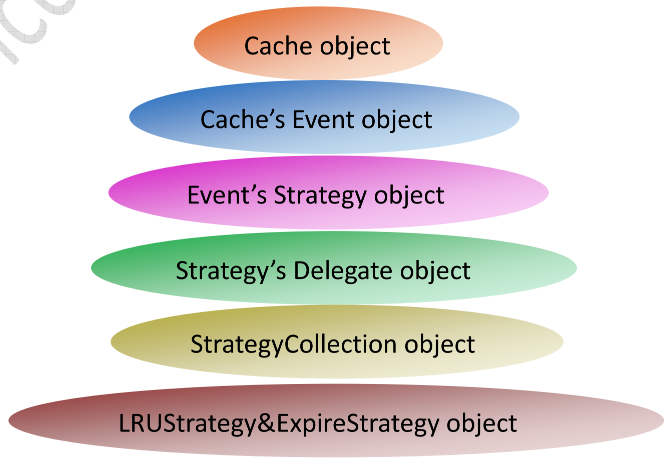
these method will be operating directly on the datas of a cache to achieve desired strategy.
in LRUStrategy, these methods are implemented in LRU ways which in ExpireStrategy these methods are implemented in expire ways. Note: all these strategies will be invoked passively while there is any event/action in cache, no proactive operation is defined here in ExpireLRUCache, event with the expire action.

6. by _strategy.pushBack, the instances of the two strategies will be added to the instance of StrategyCollection's std::vector member – Strategies

Class method invoking chain

when invoking class method like ExpireLRUCache.add() following invoking flow will occur:(basically firing events and adding in cache's data set)

ExpireLRUCache::add equals AbstractCache::add → AbstractCache::doAdd → AbstractCache::notify(this, args) equals FIFIEvent:notify equals AbstractEvent::notify → strategy::notify equals FIFOStrategy::notify equals DefaultStrategy::notify → (*it)->notify(sender, arguments) equals Delegate::notify → (_receiverObject->*_receiverMethod)(sender, arguments) equals StrategyCollection::onAdd → for (; it != endIt; ++it) (*it)->onAdd(pSender, key) equals LRUStrategy::onAdd and ExpireStrategy::onAdd which operate on data set of the cache.


Cache object
Cache's Event object
Event's Strategy object
Strategy's Delegate object
StrategyCollection object
LRUStrategy&ExpireStrategy object

Some comments regarding above flow:

eqmcc blog.csdn.net/eqmcc

in ScanCache, there’s a method called add() which is inherited from its base class ExpireLRUCache which is inherited from AbstractCache and in which doAdd() is called.

in AbstractCache.doAdd(), will do two things: 1) update the data set of cache object; 2) firing up event and doing notify. For the AbstractCache, the event is FIFOEvent type which in turn calls its base class’ action - AbstractEvent::notify.

however, for AbstractEvent, it has a strategy for event handling – FIFOStrategy, so FIFOStrategy::notify will be called; FIFOStrategy::notify is inherited from DefaultStrategy.

for FIFOStrategy, all it’s event handling is delegated out which results in calling the delegated target method - StrategyCollection::onAdd.

for every strategy in StrategyCollection(LRUStrategy and ExpireStrategy), its onAdd method will be called which finally does the expire operation and LRU update on the cache.

About Delegate

Delegate is derived from AbstractDelegate in which defines following methods:

notify
equals
clone
disable
unwrap

Delegate has following protected data:

```
TObj* _receiverObject;
NotifyMethod _receiverMethod;
```

receiverObject is the delegator and _receiverMethod is the method that the delegator trust the delegate with and when notify comes, delegate will run the delegator’s delegated method:

```
(_receiverObject->*_receiverMethod)(arguments);
```

eqmcc@http://[illegible]n.net/eqmcc

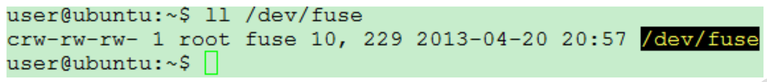
FUSE in ClamFS

there are three components in FUSE:

1. the kernel space driver
2. the user space file system
3. the user space daemon

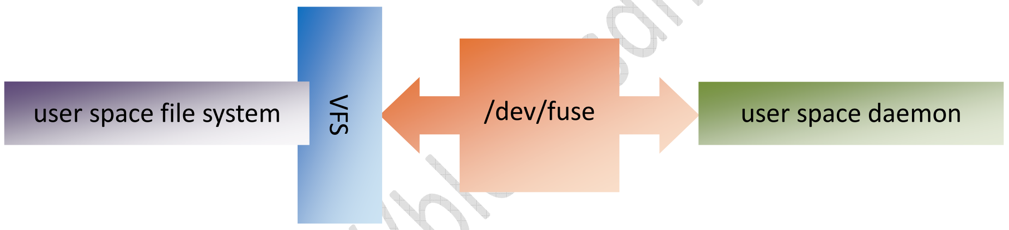
the kernel space driver

the kernel space code will register the fuse device driver into the kernel or VFS to be accurate. After kernel patched with fuse code started, the kernel will create a character device as bellow:



```
user@ubuntu:~$ ll /dev/fuse
crw-rw-rw- 1 root fuse 10, 229 2013-04-20 20:57 /dev/fuse
user@ubuntu:~$ 
```

this device will be treated as an message exchanger between the user space file system and the user space daemon:



the user space file system

there's a development library in fuse. this lib will be used to create and mount a FUSE file system in user space will the operations defined by user in fuse_operations, e.g. in clamfs:

eqmcc@http://blog.csdn.net/eqmcc

```
    clamfs_oper.getattr     = clamfs_getattr;
    clamfs_oper.fgetattr    = clamfs_fgetattr;
    clamfs_oper.access      = clamfs_access;
    clamfs_oper.readlink    = clamfs_readlink;
    clamfs_oper.opendir     = clamfs_opendir;
    clamfs_oper.readdir     = clamfs_readdir;
    clamfs_oper.releasedir  = clamfs_releasedir;
    clamfs_oper.mknod       = clamfs_mknod;
    clamfs_oper.mkdir       = clamfs_mkdir;
    clamfs_oper.symlink     = clamfs_symlink;
    clamfs_oper.unlink      = clamfs_unlink;
    clamfs_oper.rmdir       = clamfs_rmdir;
    clamfs_oper.rename      = clamfs_rename;
    clamfs_oper.link        = clamfs_link;
    clamfs_oper.chmod       = clamfs_chmod;
    clamfs_oper.chown       = clamfs_chown;
    clamfs_oper.truncate    = clamfs_truncate;
    clamfs_oper.ftruncate   = clamfs_ftruncate;
    clamfs_oper.utime       = clamfs_utime;
    clamfs_oper.create      = clamfs_create;
    clamfs_oper.open        = clamfs_open;
    clamfs_oper.read        = clamfs_read;
    clamfs_oper.write       = clamfs_write;
    clamfs_oper.statfs      = clamfs_statfs;
    clamfs_oper.release     = clamfs_release;
    clamfs_oper.fsync       = clamfs_fsync;
#ifdef HAVE_SETXATTR
    clamfs_oper.setxattr    = clamfs_setxattr;
    clamfs_oper.getxattr    = clamfs_getxattr;
    clamfs_oper.listxattr   = clamfs_listxattr;
    clamfs_oper.removexattr = clamfs_removexattr;
#endif
```

after mounting, there will be a fully functional file system exposed to any user in this OS.
when a file operation is fired, e.g.: a ls command is run against a file in user space file system, the request will be forward to VFS where, the underlying target file system of the operation will be recognized and the operation call will be forwarded to the hooked up kernel driver.
correspondingly, when a file operation is finished, the kernel will notify VFS which in turn responds back to the ls command call.

the user space daemon

the fuse lib will invoking fuse_main function which will finally start a user space daemon(fuse_session_loop) to listen for any file operation request passed from kernel. When a operation request is intercepted down from kernel, used defined file system operation function(defined in fuse_operations) will be invoked to operate the file system the way the user wants.
after user defined operation is invoked and returns, the daemon will write the results back to kernel device and which will be forwarded back to original caller.

eqmcc@http://blog.csdn.net/eqmcc