Cache on Already Mounted Filesystem

Overview

CacheFiles is a caching backend that's meant to use as a cache a directory on an already mounted filesystem of a local type (such as Ext3).

CacheFiles uses a userspace daemon to do some of the cache management - such as reaping stale nodes and culling. This is called cachefilesd and lives in /sbin.

The filesystem and data integrity of the cache are only as good as those of the filesystem providing the backing services. Note that CacheFiles does not attempt to journal anything since the journalling interfaces of the various filesystems are very specific in nature.

CacheFiles creates a misc character device - "/dev/cachefiles" - that is used to communication with the daemon. Only one thing may have this open at once, and while it is open, a cache is at least partially in existence. The daemon opens this and sends commands down it to control the cache.

CacheFiles is currently limited to a single cache.

CacheFiles attempts to maintain at least a certain percentage of free space on the filesystem, shrinking the cache by culling the objects it contains to make space if necessary - see the "Cache Culling" section. This means it can be placed on the same medium as a live set of data, and will expand to make use of spare space and automatically contract when the set of data requires more space.

Requirements

The use of CacheFiles and its daemon requires the following features to be available in the system and in the cache filesystem:

- dnotify.
- extended attributes (xattrs).
- openat() and friends.
- bmap() support on files in the filesystem (FIBMAP ioctl).
- The use of bmap() to detect a partial page at the end of the file.

It is strongly recommended that the "dir_index" option is enabled on Ext3 filesystems being used as a cache.

Configuration

The cache is configured by a script in /etc/cachefilesd.conf. These commands set up cache ready for use. The following script commands are available:

brun <N>%, bcull <N>%, bstop <N>%, frun <N>%, fcull <N>%, fstop <N>%

Configure the culling limits. Optional. See the section on culling The defaults are 7% (run), 5% (cull) and 1% (stop) respectively.

The commands beginning with a 'b' are file space (block) limits, those beginning with an 'f' are file count limits.

dir <path>

Specify the directory containing the root of the cache. Mandatory.

tag <name>

Specify a tag to FS-Cache to use in distinguishing multiple caches. Optional. The default is "CacheFiles".

debug <mask>

Specify a numeric bitmask to control debugging in the kernel module. Optional. The default is zero (all off). The following values can be OR'd into the mask to collect various information:

1	Turn on trace of function entry (_enter() macros)
2	Turn on trace of function exit (_leave() macros)
4	Turn on trace of internal debug points (_debug())

This mask can also be set through sysfs, eg:

```
echo 5 >/sys/modules/cachefiles/parameters/debug
```

Starting the Cache

The cache is started by running the daemon. The daemon opens the cache device, configures the cache and tells it to begin caching.

At that point the cache binds to fscache and the cache becomes live.

The daemon is run as follows:

```
/sbin/cachefilesd [-d]* [-s] [-n] [-f <configfile>]
```

The flags are:

`-d`
: Increase the debugging level. This can be specified multiple times and is cumulative with itself.

`-s`
: Send messages to stderr instead of syslog.

`-n`
: Don't daemonise and go into background.

`-f <configfile>`
: Use an alternative configuration file rather than the default one.

Things to Avoid

Do not mount other things within the cache as this will cause problems. The kernel module contains its own very cut-down path walking facility that ignores mountpoints, but the daemon can't avoid them.

Do not create, rename or unlink files and directories in the cache while the cache is active, as this may cause the state to become uncertain.

Renaming files in the cache might make objects appear to be other objects (the filename is part of the lookup key).

Do not change or remove the extended attributes attached to cache files by the cache as this will cause the cache state management to get confused.

Do not create files or directories in the cache, lest the cache get confused or serve incorrect data.

Do not chmod files in the cache. The module creates things with minimal permissions to prevent random users being able to access them directly.

Cache Culling

The cache may need culling occasionally to make space. This involves discarding objects from the cache that have been used less recently than anything else. Culling is based on the access time of data objects. Empty directories are culled if not in use.

Cache culling is done on the basis of the percentage of blocks and the percentage of files available in the underlying filesystem. There are six "limits":

brun, frun
: If the amount of free space and the number of available files in the cache rises above both these limits, then culling is turned off.

bcull, fcull
: If the amount of available space or the number of available files in the cache falls below either of these limits, then culling is started.

bstop, fstop
: If the amount of available space or the number of available files in the cache falls below either of these limits, then no further allocation of disk space or files is permitted until culling has raised things above these limits again.

These must be configured thusly:

```
0 <= bstop < bcull < brun < 100
0 <= fstop < fcull < frun < 100
```

Note that these are percentages of available space and available files, and do _not_ appear as 100 minus the percentage displayed by the "df" program.

The userspace daemon scans the cache to build up a table of cullable objects. These are then culled in least recently used order. A new scan of the cache is started as soon as space is made in the table. Objects will be skipped if their atimes have changed or if the kernel module says it is still using them.

Cache Structure

The CacheFiles module will create two directories in the directory it was given:

- cache/
- graveyard/

The active cache objects all reside in the first directory. The CacheFiles kernel module moves any retired or culled objects that it can't simply unlink to the graveyard from which the daemon will actually delete them.

The daemon uses dnotify to monitor the graveyard directory, and will delete anything that appears therein.

The module represents index objects as directories with the filename "I..." or "J...". Note that the "cache/" directory is itself a special index.

Data objects are represented as files if they have no children, or directories if they do. Their filenames all begin "D..." or "E...". If represented as a directory, data objects will have a file in the directory called "data" that actually holds the data.

Special objects are similar to data objects, except their filenames begin "S..." or "T...".

If an object has children, then it will be represented as a directory. Immediately in the representative directory are a collection of directories named for hash values of the child object keys with an '@' prepended. Into this directory, if possible, will be placed the representations of the child objects:

```
 /INDEX    /INDEX     /INDEX                          /DATA FILES
/=========/==========/================================/================
cache/@4a/I03nfs/@30/Ji000000000000000--fHg8hi8400
cache/@4a/I03nfs/@30/Ji000000000000000--fHg8hi8400/@75/Es0g000w...DB1ry
cache/@4a/I03nfs/@30/Ji000000000000000--fHg8hi8400/@75/Es0g000w...N22ry
cache/@4a/I03nfs/@30/Ji000000000000000--fHg8hi8400/@75/Es0g000w...FP1ry
```

If the key is so long that it exceeds NAME_MAX with the decorations added on to it, then it will be cut into pieces, the first few of which will be used to make a nest of directories, and the last one of which will be the objects inside the last directory. The names of the intermediate directories will have '+' prepended:

```
J1223/@23/+xy...z/+kl...m/Epqr
```

Note that keys are raw data, and not only may they exceed NAME_MAX in size, they may also contain things like '/' and NUL characters, and so they may not be suitable for turning directly into a filename.

To handle this, CacheFiles will use a suitably printable filename directly and "base-64" encode ones that aren't directly suitable. The two versions of object filenames indicate the encoding:

OBJECT TYPE	PRINTABLE	ENCODED
Index	"I..."	"J..."
Data	"D..."	"E..."
Special	"S..."	"T..."

Intermediate directories are always "@" or "+" as appropriate.

Each object in the cache has an extended attribute label that holds the object type ID (required to distinguish special objects) and the auxiliary data from the netfs. The latter is used to detect stale objects in the cache and update or retire them.

Note that CacheFiles will erase from the cache any file it doesn't recognise or any file of an incorrect type (such as a FIFO file or a device file).

Security Model and SELinux

CacheFiles is implemented to deal properly with the LSM security features of the Linux kernel and the SELinux facility.

One of the problems that CacheFiles faces is that it is generally acting on behalf of a process, and running in that process's context, and that includes a security context that is not appropriate for accessing the cache - either because the files in the cache are inaccessible to that process, or because if the process creates a file in the cache, that file may be inaccessible to other processes.

The way CacheFiles works is to temporarily change the security context (fsuid, fsgid and actor security label) that the process acts as - without changing the security context of the process when it the target of an operation performed by some other process (so signalling and suchlike still work correctly).

When the CacheFiles module is asked to bind to its cache, it:

1. Finds the security label attached to the root cache directory and uses that as the security label with which it will create files. By default, this is:

   ```
   cachefiles_var_t
   ```

2. Finds the security label of the process which issued the bind request (presumed to be the cachefilesd daemon), which by default will be:

   ```
   cachefilesd_t
   ```

   and asks LSM to supply a security ID as which it should act given the daemon's label. By default, this will be:

   ```
   cachefiles_kernel_t
   ```

SELinux transitions the daemon's security ID to the module's security ID based on a rule of this form in the policy:

```
type_transition <daemon's-ID> kernel_t : process <module's-ID>;
```

For instance:

```
type_transition cachefilesd_t kernel_t : process cachefiles_kernel_t;
```

The module's security ID gives it permission to create, move and remove files and directories in the cache, to find and access directories and files in the cache, to set and access extended attributes on cache objects, and to read and write files in the cache.

The daemon's security ID gives it only a very restricted set of permissions: it may scan directories, stat files and erase files and directories. It may not read or write files in the cache, and so it is precluded from accessing the data cached therein; nor is it permitted to create new files in the cache.

There are policy source files available in:

https://people.redhat.com/~dhowells/fscache/cachefilesd-0.8.tar.bz2

and later versions. In that tarball, see the files:

```
cachefilesd.te
cachefilesd.fc
cachefilesd.if
```

They are built and installed directly by the RPM.

If a non-RPM based system is being used, then copy the above files to their own directory and run:

```
make -f /usr/share/selinux/devel/Makefile
semodule -i cachefilesd.pp
```

You will need checkpolicy and selinux-policy-devel installed prior to the build.

By default, the cache is located in /var/fscache, but if it is desirable that it should be elsewhere, than either the above policy files must be altered, or an auxiliary policy must be installed to label the alternate location of the cache.

For instructions on how to add an auxiliary policy to enable the cache to be located elsewhere when SELinux is in enforcing mode, please see:

```
/usr/share/doc/cachefilesd-*/move-cache.txt
```

When the cachefilesd rpm is installed; alternatively, the document can be found in the sources.

A Note on Security

CacheFiles makes use of the split security in the task_struct. It allocates its own task_security structure, and redirects current->cred to point to it when it acts on behalf of another process, in that process's context.

The reason it does this is that it calls vfs_mkdir() and suchlike rather than bypassing security and calling inode ops directly. Therefore the VFS and LSM may deny the CacheFiles access to the cache data because under some circumstances the caching code is running in the security context of whatever process issued the original syscall on the netfs.

Furthermore, should CacheFiles create a file or directory, the security parameters with that object is created (UID, GID, security label) would be derived from that process that issued the system call, thus potentially preventing other processes from accessing the cache - including CacheFiles's cache management daemon (cachefilesd).

What is required is to temporarily override the security of the process that issued the system call. We can't, however, just do an in-place change of the security data as that affects the process as an object, not just as a subject. This means it may lose signals or ptrace events for example, and affects what the process looks like in /proc.

So CacheFiles makes use of a logical split in the security between the objective security (task->real_cred) and the subjective security (task->cred). The objective security holds the intrinsic security properties of a process and is never overridden. This is what appears in /proc, and is what is used when a process is the target of an operation by some other process (SIGKILL for example).

The subjective security holds the active security properties of a process, and may be overridden. This is not seen externally, and is used whan a process acts upon another object, for example SIGKILLing another process or opening a file.

LSM hooks exist that allow SELinux (or Smack or whatever) to reject a request for CacheFiles to run in a context of a specific security label, or to create files and directories with another security label.

Statistical Information

If FS-Cache is compiled with the following option enabled:

```
CONFIG_CACHEFILES_HISTOGRAM=y
```

then it will gather certain statistics and display them through a proc file.

/proc/fs/cachefiles/histogram

```
cat /proc/fs/cachefiles/histogram
JIFS  SECS  LOOKUPS   MKDIRS    CREATES
===== ===== ========= ========= =========
```

This shows the breakdown of the number of times each amount of time between 0 jiffies and HZ-1 jiffies a variety of tasks took to run. The columns are as follows:

COLUMN	TIME MEASUREMENT
LOOKUPS	Length of time to perform a lookup on the backing fs
MKDIRS	Length of time to perform a mkdir on the backing fs
CREATES	Length of time to perform a create on the backing fs

Each row shows the number of events that took a particular range of times. Each step is 1 jiffy in size. The JIFS column indicates the particular jiffy range covered, and the SECS field the equivalent number of seconds.

Debugging

If CONFIG_CACHEFILES_DEBUG is enabled, the CacheFiles facility can have runtime debugging enabled by adjusting the value in:

```
/sys/module/cachefiles/parameters/debug
```

This is a bitmask of debugging streams to enable:

BIT	VALUE	STREAM	POINT
0	1	General	Function entry trace
1	2		Function exit trace
2	4		General

The appropriate set of values should be OR'd together and the result written to the control file. For example:

```
echo $((1|4|8)) >/sys/module/cachefiles/parameters/debug
```

will turn on all function entry debugging.