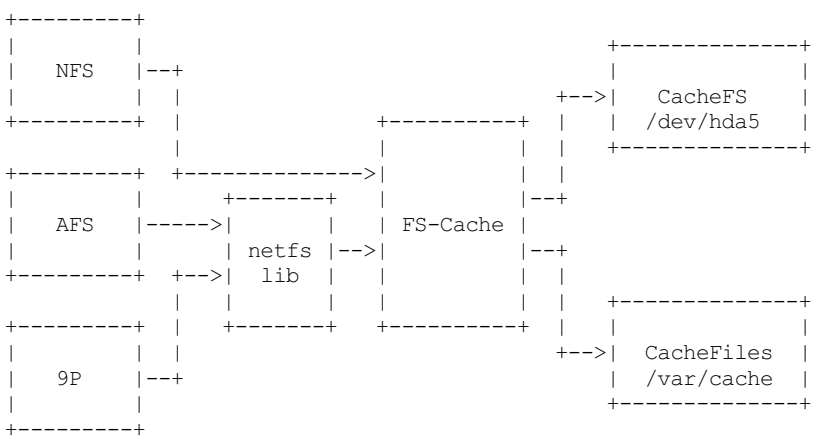
General Filesystem Caching

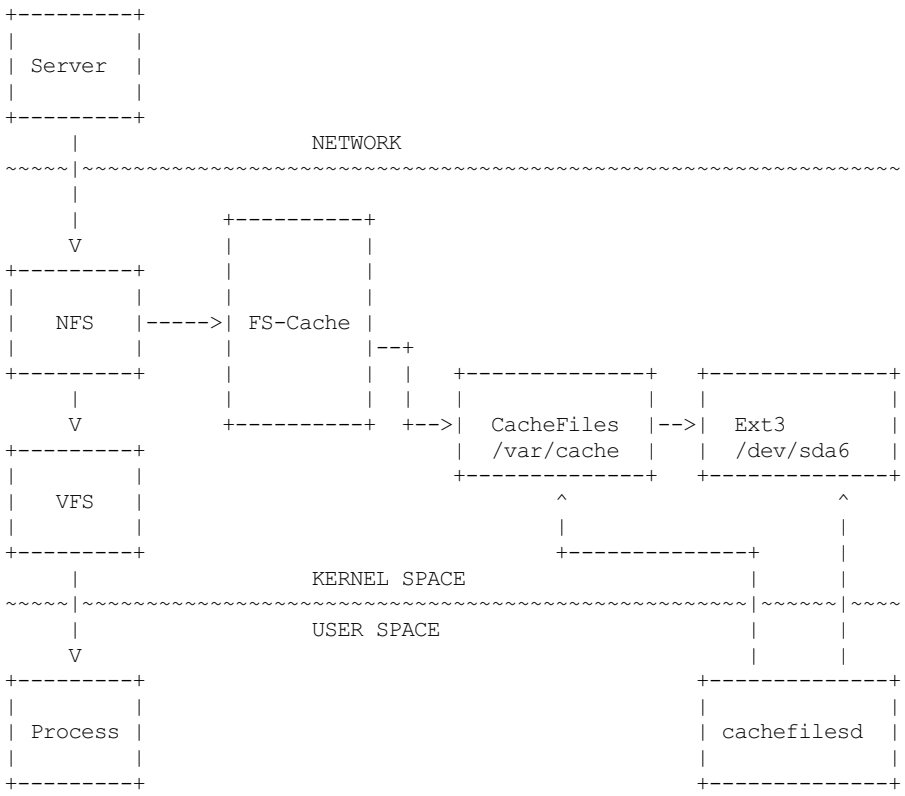
Overview

This facility is a general purpose cache for network filesystems, though it could be used for caching other things such as ISO9660 filesystems too.

FS-Cache mediates between cache backends (such as CacheFiles) and network filesystems:



Or to look at it another way, FS-Cache is a module that provides a caching facility to a network filesystem such that the cache is transparent to the user:



FS-Cache does not follow the idea of completely loading every netfs file opened in its entirety into a cache before permitting it to be accessed and then serving the pages out of that cache rather than the netfs inode because:

1. It must be practical to operate without a cache.
2. The size of any accessible file must not be limited to the size of the cache.
3. The combined size of all opened files (this includes mapped libraries) must not be limited to the size of the cache.
4. The user should not be forced to download an entire file just to do a one-off access of a small portion of it (such as might be done with the "file" program).

It instead serves the cache out in chunks as and when requested by the netfs using it.

FS-Cache provides the following facilities:

- More than one cache can be used at once. Caches can be selected explicitly by use of tags.

- Caches can be added / removed at any time, even whilst being accessed.
- The netfs is provided with an interface that allows either party to withdraw caching facilities from a file (required for (2)).
- The interface to the netfs returns as few errors as possible, preferring rather to let the netfs remain oblivious.
- There are three types of cookie: cache, volume and data file cookies. Cache cookies represent the cache as a whole and are not normally visible to the netfs; the netfs gets a volume cookie to represent a collection of files (typically something that a netfs would get for a superblock); and data file cookies are used to cache data (something that would be got for an inode).
- Volumes are matched using a key. This is a printable string that is used to encode all the information that might be needed to distinguish one superblock, say, from another. This would be a compound of things like cell name or server address, volume name or share path. It must be a valid pathname.
- Cookies are matched using a key. This is a binary blob and is used to represent the object within a volume (so the volume key need not form part of the blob). This might include things like an inode number and uniquifier or a file handle.
- Cookie resources are set up and pinned by marking the cookie in-use. This prevents the backing resources from being culled. Timed garbage collection is employed to eliminate cookies that haven't been used for a short while, thereby reducing resource overload. This is intended to be used when a file is opened or closed.

  A cookie can be marked in-use multiple times simultaneously; each mark must be unused.
- Begin/end access functions are provided to delay cache withdrawal for the duration of an operation and prevent structs from being freed whilst we're looking at them.
- Data I/O is done by asynchronous DIO to/from a buffer described by the netfs using an iov_iter.
- An invalidation facility is available to discard data from the cache and to deal with I/O that's in progress that is accessing old data.
- Cookies can be "retired" upon release, thereby causing the object to be removed from the cache.

The netfs API to FS-Cache can be found in:

    Documentation/filesystems/caching/netfs-api.rst

The cache backend API to FS-Cache can be found in:

    Documentation/filesystems/caching/backend-api.rst

Statistical Information

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

```
CONFIG_FSCACHE_STATS=y
```

then it will gather certain statistics and display them through:

    /proc/fs/fscache/stats

This shows counts of a number of events that can happen in FS-Cache:

CLASS	EVENT	MEANING
Cookies	n=N	Number of data storage cookies allocated
	v=N	Number of volume index cookies allocated
	vcol=N	Number of volume index key collisions
	voom=N	Number of OOM events when allocating volume cookies
Acquire	n=N	Number of acquire cookie requests seen
	ok=N	Number of acq reqs succeeded
	oom=N	Number of acq reqs failed on ENOMEM
LRU	n=N	Number of cookies currently on the LRU
	exp=N	Number of cookies expired off of the LRU
	rmv=N	Number of cookies removed from the LRU
	drp=N	Number of LRU'd cookies relinquished/withdrawn
	at=N	Time till next LRU cull (jiffies)
Invals	n=N	Number of invalidations
Updates	n=N	Number of update cookie requests seen
	rsz=N	Number of resize requests
	rsn=N	Number of skipped resize requests

CLASS	EVENT	MEANING
Relinqs	n=N	Number of relinquish cookie requests seen
	rtr=N	Number of rlq reqs with retire=true
	drop=N	Number of cookies no longer blocking re-acquisition
NoSpace	nwr=N	Number of write requests refused due to lack of space
	ncr=N	Number of create requests refused due to lack of space
	cull=N	Number of objects culled to make space
IO	rd=N	Number of read operations in the cache
	wr=N	Number of write operations in the cache

Netfslib will also add some stats counters of its own.

Cache List

FS-Cache provides a list of cache cookies:

/proc/fs/fscache/cookies

This will look something like:

```
# cat /proc/fs/fscache/caches
CACHE    REF   VOLS  OBJS  ACCES S NAME
======== ===== ===== ===== ===== = ===============
00000001     2     1  2123     1 A default
```

where the columns are:

COLUMN	DESCRIPTION
CACHE	Cache cookie debug ID (also appears in traces)
REF	Number of references on the cache cookie
VOLS	Number of volumes cookies in this cache
OBJS	Number of cache objects in use
ACCES	Number of accesses pinning the cache
S	State
NAME	Name of the cache.

The state can be (-) Inactive, (P)reparing, (A)ctive, (E)rror or (W)ithdrawing.

Volume List

FS-Cache provides a list of volume cookies:

/proc/fs/fscache/volumes

This will look something like:

```
VOLUME   REF   nCOOK ACC FL CACHE           KEY
======== ===== ===== === == =============== ================
00000001    55    54   1 00 default         afs,example.com,100058
```

where the columns are:

COLUMN	DESCRIPTION
VOLUME	The volume cookie debug ID (also appears in traces)
REF	Number of references on the volume cookie
nCOOK	Number of cookies in the volume
ACC	Number of accesses pinning the cache
FL	Flags on the volume cookie
CACHE	Name of the cache or "-"
KEY	The indexing key for the volume

Cookie List

FS-Cache provides a list of cookies:

/proc/fs/fscache/cookies

This will look something like:

```
# head /proc/fs/fscache/cookies
COOKIE   VOLUME   REF ACT ACC S FL DEF
======== ======== === === === = == ================
00000435 00000001   1   0  -1 - 08 0000000201d0800700000000000000000, 0000000000000000
00000436 00000001   1   0  -1 - 00 0000005601d0800800000000000000000, 0000000000000051
00000437 00000001   1   0  -1 - 08 00023b3001d0823f0000000000000000, 0000000000000000
00000438 00000001   1   0  -1 - 08 0000005801d0807b0000000000000000, 0000000000000000
00000439 00000001   1   0  -1 - 08 00023b3201d080a10000000000000000, 0000000000000000
0000043a 00000001   1   0  -1 - 08 00023b3401d080a30000000000000000, 0000000000000000
0000043b 00000001   1   0  -1 - 08 00023b3601d080b30000000000000000, 0000000000000000
0000043c 00000001   1   0  -1 - 08 00023b3801d080b40000000000000000, 0000000000000000
```

where the columns are:

COLUMN	DESCRIPTION
COOKIE	The cookie debug ID (also appears in traces)
VOLUME	The parent volume cookie debug ID
REF	Number of references on the volume cookie
ACT	Number of times the cookie is marked for in use
ACC	Number of access pins in the cookie
S	State of the cookie
FL	Flags on the cookie
DEF	Key, auxiliary data

Debugging

If CONFIG_FSCACHE_DEBUG is enabled, the FS-Cache facility can have runtime debugging enabled by adjusting the value in:

```
/sys/module/fscache/parameters/debug
```

This is a bitmask of debugging streams to enable:

BIT	VALUE	STREAM	POINT
0	1	Cache management	Function entry trace
1	2		Function exit trace
2	4		General
3	8	Cookie management	Function entry trace
4	16		Function exit trace
5	32		General
6-8			(Not used)
9	512	I/O operation management	Function entry trace
10	1024		Function exit trace
11	2048		General

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

```
echo $((1|8|512)) >/sys/module/fscache/parameters/debug
```

will turn on all function entry debugging.