==========================
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 CacheFS) and network filesystems:

```
        +---------+
        |         |                        +--------------+
        |   NFS   |--+                     |              |
        |         |  |                 +-->|   CacheFS    |
        +---------+  |   +----------+  |   |  /dev/hda5   |
                     |   |          |  |   +--------------+
        +---------+  +-->|          |  |
        |         |      |          |--+
        |   AFS   |----->| FS-Cache |
        |         |      |          |--+
        +---------+  +-->|          |  |
                     |   |          |  |   +--------------+
        +---------+  |   +----------+  |   |              |
        |         |  |                 +-->|  CacheFiles  |
        |  ISOFS  |--+                     |  /var/cache  |
        |         |                        +--------------+
        +---------+
```

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:

```
        +---------+
        |         |
        | Server  |
        |         |
        +---------+
             |                  NETWORK
        ~~~~~|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
             |
             |           +----------+
             V           |          |
        +---------+      |          |
        |         |      |          |
        |   NFS   |----->| FS-Cache |
        |         |      |          |--+
        +---------+      |          |  |   +--------------+   +--------------+
             |           |          |  |   |              |   |              |
             V           +----------+  +-->|  CacheFiles  |-->|  Ext3        |
        +---------+                        |  /var/cache  |   |  /dev/sda6   |
        |         |                        +--------------+   +--------------+
        |   VFS   |                                ^                     ^
```

```
    |         |                                 |
    +---------+                                 +--------------+
         |                 KERNEL SPACE                        |      |
~~~~~|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|~~~~~~|~~~~
         |                 USER SPACE                          |      |
         V                                                     |      |
    +---------+                                         +--------------+
    |         |                                         |              |
    | Process |                                         | cachefilesd  |
    |         |                                         |              |
    +---------+                                         +--------------+
```

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 PAGE_SIZE chunks as and when requested by the netfs('s) using it.

FS-Cache provides the following facilities:

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

(2) Caches can be added / removed at any time.

(3) The netfs is provided with an interface that allows either party to withdraw caching facilities from a file (required for (2)).

(4) The interface to the netfs returns as few errors as possible, preferring rather to let the netfs remain oblivious.

(5) Cookies are used to represent indices, files and other objects to the netfs. The simplest cookie is just a NULL pointer - indicating nothing cached there.

(6) The netfs is allowed to propose - dynamically - any index hierarchy it desires, though it must be aware that the index search function is recursive, stack space is limited, and indices can only be children of indices.

(7) Data I/O is done direct to and from the netfs's pages. The netfs

indicates that page A is at index B of the data-file represented by cookie C, and that it should be read or written. The cache backend may or may not start I/O on that page, but if it does, a netfs callback will be invoked to indicate completion. The I/O may be either synchronous or asynchronous.

(8) Cookies can be "retired" upon release. At this point FS-Cache will mark them as obsolete and the index hierarchy rooted at that point will get recycled.

(9) The netfs provides a "match" function for index searches. In addition to saying whether a match was made or not, this can also specify that an entry should be updated or deleted.

(10) As much as possible is done asynchronously.

FS-Cache maintains a virtual indexing tree in which all indices, files, objects and pages are kept. Bits of this tree may actually reside in one or more caches.

```
                                           FSDEF
                                             |
                        +------------------------------------+
                        |                                    |
                       NFS                                  AFS
                        |                                    |
           +--------------------------+                +-----------+
           |                          |                |           |
        homedir                     mirror          afs.org   redhat.com
           |                          |                            |
     +------------+           +---------------+              +----------+
     |            |           |               |              |          |
   00001        00002       00007           00125        vol00001   vol00002
     |            |           |               |                         |
 +---+---+     +-----+      +---+      +------+------+            +-----+----+
 |   |   |     |     |      |   |      |      |      |            |     |    |
PG0 PG1 PG2   PG0  XATTR   PG0 PG1   DIRENT DIRENT DIRENT        R/W   R/O  Bak
                     |                                            |
                    PG0                                       +-------+
                                                              |       |
                                                            00001   00003
                                                              |
                                                          +---+---+
                                                          |   |   |
                                                         PG0 PG1 PG2
```

In the example above, you can see two netfs's being backed: NFS and AFS. These have different index hierarchies:

(*) The NFS primary index contains per-server indices. Each server index is indexed by NFS file handles to get data file objects. Each data file objects can have an array of pages, but may also have further child objects, such as extended attributes and directory entries. Extended attribute objects themselves have page-array contents.

(*) The AFS primary index contains per-cell indices. Each cell index contains per-logical-volume indices. Each of volume index contains up to three indices for the read-write, read-only and backup mirrors of those volumes. Each of these contains vnode data file objects, each of which contains an array of pages.

The very top index is the FS-Cache master index in which individual netfs's have entries.

Any index object may reside in more than one cache, provided it only has index children. Any index with non-index object children will be assumed to only reside in one cache.

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

    Documentation/filesystems/caching/netfs-api.txt

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

    Documentation/filesystems/caching/backend-api.txt

A description of the internal representations and object state machine can be found in:

    Documentation/filesystems/caching/object.txt

STATISTICAL INFORMATION

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

```
CONFIG_FSCACHE_STATS=y
CONFIG_FSCACHE_HISTOGRAM=y
```

then it will gather certain statistics and display them through a number of proc files.

(*) /proc/fs/fscache/stats

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

CLASS	EVENT	MEANING
Cookies	idx=N	Number of index cookies allocated
	dat=N	Number of data storage cookies allocated
	spc=N	Number of special cookies allocated
Objects	alc=N	Number of objects allocated
	nal=N	Number of object allocation failures
	avl=N	Number of objects that reached the available state
	ded=N	Number of objects that reached the dead state
ChkAux	non=N	Number of objects that didn't have a coherency check
	ok=N	Number of objects that passed a coherency check
	upd=N	Number of objects that needed a coherency data update

```
                obs=N   Number of objects that were declared obsolete
        Pages   mrk=N   Number of pages marked as being cached
                unc=N   Number of uncache page requests seen
        Acquire n=N     Number of acquire cookie requests seen
                nul=N   Number of acq reqs given a NULL parent
                noc=N   Number of acq reqs rejected due to no cache available
                ok=N    Number of acq reqs succeeded
                nbf=N   Number of acq reqs rejected due to error
                oom=N   Number of acq reqs failed on ENOMEM
        Lookups n=N     Number of lookup calls made on cache backends
                neg=N   Number of negative lookups made
                pos=N   Number of positive lookups made
                crt=N   Number of objects created by lookup
                tmo=N   Number of lookups timed out and requeued
        Updates n=N     Number of update cookie requests seen
                nul=N   Number of upd reqs given a NULL parent
                run=N   Number of upd reqs granted CPU time
        Relinqs n=N     Number of relinquish cookie requests seen
                nul=N   Number of rlq reqs given a NULL parent
                wcr=N   Number of rlq reqs waited on completion of creation
        AttrChg n=N     Number of attribute changed requests seen
                ok=N    Number of attr changed requests queued
                nbf=N   Number of attr changed rejected -ENOBUFS
                oom=N   Number of attr changed failed -ENOMEM
                run=N   Number of attr changed ops given CPU time
        Allocs  n=N     Number of allocation requests seen
                ok=N    Number of successful alloc reqs
                wt=N    Number of alloc reqs that waited on lookup completion
                nbf=N   Number of alloc reqs rejected -ENOBUFS
                int=N   Number of alloc reqs aborted -ERESTARTSYS
                ops=N   Number of alloc reqs submitted
                owt=N   Number of alloc reqs waited for CPU time
                abt=N   Number of alloc reqs aborted due to object death
        Retrvls n=N     Number of retrieval (read) requests seen
                ok=N    Number of successful retr reqs
                wt=N    Number of retr reqs that waited on lookup completion
                nod=N   Number of retr reqs returned -ENODATA
                nbf=N   Number of retr reqs rejected -ENOBUFS
                int=N   Number of retr reqs aborted -ERESTARTSYS
                oom=N   Number of retr reqs failed -ENOMEM
                ops=N   Number of retr reqs submitted
                owt=N   Number of retr reqs waited for CPU time
                abt=N   Number of retr reqs aborted due to object death
        Stores  n=N     Number of storage (write) requests seen
                ok=N    Number of successful store reqs
                agn=N   Number of store reqs on a page already pending storage
                nbf=N   Number of store reqs rejected -ENOBUFS
                oom=N   Number of store reqs failed -ENOMEM
                ops=N   Number of store reqs submitted
                run=N   Number of store reqs granted CPU time
                pgs=N   Number of pages given store req processing time
                rxd=N   Number of store reqs deleted from tracking tree
                olm=N   Number of store reqs over store limit
        VmScan  nos=N   Number of release reqs against pages with no pending
store
                gon=N   Number of release reqs against pages stored by time lock
```

```
granted
                bsy=N   Number of release reqs ignored due to in-progress store
                can=N   Number of page stores cancelled due to release req
        Ops     pend=N  Number of times async ops added to pending queues
                run=N   Number of times async ops given CPU time
                enq=N   Number of times async ops queued for processing
                can=N   Number of async ops cancelled
                rej=N   Number of async ops rejected due to object lookup/create
failure
                dfr=N   Number of async ops queued for deferred release
                rel=N   Number of async ops released
                gc=N    Number of deferred-release async ops garbage collected
        CacheOp alo=N   Number of in-progress alloc_object() cache ops
                luo=N   Number of in-progress lookup_object() cache ops
                luc=N   Number of in-progress lookup_complete() cache ops
                gro=N   Number of in-progress grab_object() cache ops
                upo=N   Number of in-progress update_object() cache ops
                dro=N   Number of in-progress drop_object() cache ops
                pto=N   Number of in-progress put_object() cache ops
                syn=N   Number of in-progress sync_cache() cache ops
                atc=N   Number of in-progress attr_changed() cache ops
                rap=N   Number of in-progress read_or_alloc_page() cache ops
                ras=N   Number of in-progress read_or_alloc_pages() cache ops
                alp=N   Number of in-progress allocate_page() cache ops
                als=N   Number of in-progress allocate_pages() cache ops
                wrp=N   Number of in-progress write_page() cache ops
                ucp=N   Number of in-progress uncache_page() cache ops
                dsp=N   Number of in-progress dissociate_pages() cache ops


 (*) /proc/fs/fscache/histogram

        cat /proc/fs/fscache/histogram
        JIFS  SECS  OBJ INST  OP RUNS   OBJ RUNS  RETRV DLY RETRIEVLS
        ===== ===== ========= ========= ========= ========= =========

     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
        =======         =======================================================
        OBJ INST        Length of time to instantiate an object
        OP RUNS         Length of time a call to process an operation took
        OBJ RUNS        Length of time a call to process an object event took
        RETRV DLY       Time between an requesting a read and lookup completing
        RETRIEVLS       Time between beginning and end of a retrieval

     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.


===========
OBJECT LIST
===========
```

If CONFIG_FSCACHE_OBJECT_LIST is enabled, the FS-Cache facility will maintain a list of all the objects currently allocated and allow them to be viewed through:

```
        /proc/fs/fscache/objects
```

This will look something like:

```
        [root@andromeda ~]# head /proc/fs/fscache/objects
        OBJECT   PARENT   STAT CHLDN OPS OOP IPR EX READS EM EV F S |
NETFS_COOKIE_DEF TY FL NETFS_DATA       OBJECT_KEY, AUX_DATA
        ======== ======== ==== ===== === === === == ===== == == = = |
================ == == ================ ================
           17e4b        2 ACTV     0   0   0   0  0     0 7b  4 0 8 | NFS.fh
      DT  0 ffff88001dd82820
010006017edcf8bbc93b43298fdfbe71e50b57b13a172c0117f38472,
e567634700000000000000000000000063f2404a000000000000000000000000c903000000000000
0000000063f2404a
           1693a        2 ACTV     0   0   0   0  0     0 7b  4 0 8 | NFS.fh
      DT  0 ffff88002db23380
010006017edcf8bbc93b43298fdfbe71e50b57b1e0162c01a2df0ea6,
420ebc4a000000000000000000000000420ebc4a000000000000000000000000e180100000000000
00000000420ebc4a
```

where the first set of columns before the '|' describe the object:

```
        COLUMN  DESCRIPTION
        ======= ===============================================================
        OBJECT  Object debugging ID (appears as OBJ%x in some debug messages)
        PARENT  Debugging ID of parent object
        STAT    Object state
        CHLDN   Number of child objects of this object
        OPS     Number of outstanding operations on this object
        OOP     Number of outstanding child object management operations
        IPR
        EX      Number of outstanding exclusive operations
        READS   Number of outstanding read operations
        EM      Object's event mask
        EV      Events raised on this object
        F       Object flags
        S       Object slow-work work item flags
```

and the second set of columns describe the object's cookie, if present:

```
        COLUMN          DESCRIPTION
        =============== =======================================================
        NETFS_COOKIE_DEF Name of netfs cookie definition
        TY              Cookie type (IX - index, DT - data, hex - special)
        FL              Cookie flags
        NETFS_DATA      Netfs private data stored in the cookie
        OBJECT_KEY      Object key      } 1 column, with separating comma
        AUX_DATA        Object aux data } presence may be configured
```

The data shown may be filtered by attaching the a key to an appropriate keyring before viewing the file. Something like:

```
        keyctl add user fscache:objlist <restrictions> @s
```

where <restrictions> are a selection of the following letters:

```
K       Show hexdump of object key (don't show if not given)
A       Show hexdump of object aux data (don't show if not given)
```

and the following paired letters:

```
C       Show objects that have a cookie
c       Show objects that don't have a cookie
B       Show objects that are busy
b       Show objects that aren't busy
W       Show objects that have pending writes
w       Show objects that don't have pending writes
R       Show objects that have outstanding reads
r       Show objects that don't have outstanding reads
S       Show objects that have slow work queued
s       Show objects that don't have slow work queued
```

If neither side of a letter pair is given, then both are implied. For example:

```
keyctl add user fscache:objlist KB @s
```

shows objects that are busy, and lists their object keys, but does not dump their auxiliary data. It also implies "CcWwRrSs", but as 'B' is given, 'b' is not implied.

By default all objects and all fields will be shown.

=========
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	64	Page handling	Function entry trace
7	128		Function exit trace
8	256		General
9	512	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|64)) >/sys/module/fscache/parameters/debug
```

will turn on all function entry debugging.