ASYNCHRONOUS OPERATIONS HANDLING

By: David Howells <dhowells@redhat.com>

Contents:

OVERVIEW

FS-Cache has an asynchronous operations handling facility that it uses for its data storage and retrieval routines. Its operations are represented by fscache_operation structs, though these are usually embedded into some other structure.

This facility is available to and expected to be be used by the cache backends, and FS-Cache will create operations and pass them off to the appropriate cache backend for completion.

To make use of this facility, <linux/fscache-cache.h> should be #included.

OPERATION RECORD INITIALISATION

An operation is recorded in an fscache_operation struct:

```
	struct fscache_operation {
		union {
			struct work_struct fast_work;
			struct slow_work slow_work;
		};
		unsigned long		flags;
		fscache_operation_processor_t processor;
		...
	};
```

Someone wanting to issue an operation should allocate something with this struct embedded in it. They should initialise it by calling:

```
	void fscache_operation_init(struct fscache_operation *op,
				    fscache_operation_release_t release);
```

with the operation to be initialised and the release function to use.

The op->flags parameter should be set to indicate the CPU time provision and the exclusivity (see the Parameters section).

The op->fast_work, op->slow_work and op->processor flags should be set as appropriate for the CPU time provision (see the Parameters section).

FSCACHE_OP_WAITING may be set in op->flags prior to each submission of the operation and waited for afterwards.

==========
PARAMETERS
==========

There are a number of parameters that can be set in the operation record's flag parameter. There are three options for the provision of CPU time in these operations:

(1) The operation may be done synchronously (FSCACHE_OP_MYTHREAD). A thread may decide it wants to handle an operation itself without deferring it to another thread.

This is, for example, used in read operations for calling readpages() on the backing filesystem in CacheFiles. Although readpages() does an asynchronous data fetch, the determination of whether pages exist is done synchronously - and the netfs does not proceed until this has been determined.

If this option is to be used, FSCACHE_OP_WAITING must be set in op->flags before submitting the operation, and the operating thread must wait for it to be cleared before proceeding:

```
		wait_on_bit(&op->flags, FSCACHE_OP_WAITING,
			    fscache_wait_bit, TASK_UNINTERRUPTIBLE);
```

(2) The operation may be fast asynchronous (FSCACHE_OP_FAST), in which case it will be given to keventd to process. Such an operation is not permitted to sleep on I/O.

This is, for example, used by CacheFiles to copy data from a backing fs page to a netfs page after the backing fs has read the page in.

If this option is used, op->fast_work and op->processor must be initialised before submitting the operation:

```
		INIT_WORK(&op->fast_work, do_some_work);
```

(3) The operation may be slow asynchronous (FSCACHE_OP_SLOW), in which case it will be given to the slow work facility to process. Such an operation is permitted to sleep on I/O.

```
     This is, for example, used by FS-Cache to handle background writes of
     pages that have just been fetched from a remote server.

     If this option is used, op->slow_work and op->processor must be
     initialised before submitting the operation:

		fscache_operation_init_slow(op, processor)


Furthermore, operations may be one of two types:

 (1) Exclusive (FSCACHE_OP_EXCLUSIVE).  Operations of this type may not run in
     conjunction with any other operation on the object being operated upon.

     An example of this is the attribute change operation, in which the file
     being written to may need truncation.

 (2) Shareable.  Operations of this type may be running simultaneously.  It's
     up to the operation implementation to prevent interference between other
     operations running at the same time.


=========
PROCEDURE
=========

Operations are used through the following procedure:

 (1) The submitting thread must allocate the operation and initialise it
     itself.  Normally this would be part of a more specific structure with the
     generic op embedded within.

 (2) The submitting thread must then submit the operation for processing using
     one of the following two functions:

	int fscache_submit_op(struct fscache_object *object,
			      struct fscache_operation *op);

	int fscache_submit_exclusive_op(struct fscache_object *object,
					struct fscache_operation *op);

     The first function should be used to submit non-exclusive ops and the
     second to submit exclusive ones.  The caller must still set the
     FSCACHE_OP_EXCLUSIVE flag.

     If successful, both functions will assign the operation to the specified
     object and return 0.  -ENOBUFS will be returned if the object specified is
     permanently unavailable.

     The operation manager will defer operations on an object that is still
     undergoing lookup or creation.  The operation will also be deferred if an
     operation of conflicting exclusivity is in progress on the object.

     If the operation is asynchronous, the manager will retain a reference to
     it, so the caller should put their reference to it by passing it to:
```

```
	void fscache_put_operation(struct fscache_operation *op);
```

 (3) If the submitting thread wants to do the work itself, and has marked the
     operation with FSCACHE_OP_MYTHREAD, then it should monitor
     FSCACHE_OP_WAITING as described above and check the state of the object if
     necessary (the object might have died whilst the thread was waiting).

     When it has finished doing its processing, it should call
     fscache_put_operation() on it.

 (4) The operation holds an effective lock upon the object, preventing other
     exclusive ops conflicting until it is released. The operation can be
     enqueued for further immediate asynchronous processing by adjusting the
     CPU time provisioning option if necessary, eg:

```
	op->flags &= ~FSCACHE_OP_TYPE;
	op->flags |= ~FSCACHE_OP_FAST;
```

     and calling:

```
	void fscache_enqueue_operation(struct fscache_operation *op)
```

     This can be used to allow other things to have use of the worker thread
     pools.

ASYNCHRONOUS CALLBACK

When used in asynchronous mode, the worker thread pool will invoke the
processor method with a pointer to the operation. This should then get at the
container struct by using container_of():

```
	static void fscache_write_op(struct fscache_operation *_op)
	{
		struct fscache_storage *op =
			container_of(_op, struct fscache_storage, op);
	...
	}
```

The caller holds a reference on the operation, and will invoke
fscache_put_operation() when the processor function returns. The processor
function is at liberty to call fscache_enqueue_operation() or to take extra
references.