Fiemap Ioctl

The fiemap ioctl is an efficient method for userspace to get file
extent mappings. Instead of block-by-block mapping (such as bmap), fiemap
returns a list of extents.

Request Basics

A fiemap request is encoded within struct fiemap:

```
struct fiemap {
        __u64   fm_start;         /* logical offset (inclusive) at
                                   * which to start mapping (in) */
        __u64   fm_length;        /* logical length of mapping which
                                   * userspace cares about (in) */
        __u32   fm_flags;         /* FIEMAP_FLAG_* flags for request (in/out) */
        __u32   fm_mapped_extents; /* number of extents that were
                                    * mapped (out) */
        __u32   fm_extent_count; /* size of fm_extents array (in) */
        __u32   fm_reserved;
        struct fiemap_extent fm_extents[0]; /* array of mapped extents (out) */
};
```

fm_start, and fm_length specify the logical range within the file
which the process would like mappings for. Extents returned mirror
those on disk - that is, the logical offset of the 1st returned extent
may start before fm_start, and the range covered by the last returned
extent may end after fm_length. All offsets and lengths are in bytes.

Certain flags to modify the way in which mappings are looked up can be
set in fm_flags. If the kernel doesn't understand some particular
flags, it will return EBADR and the contents of fm_flags will contain
the set of flags which caused the error. If the kernel is compatible
with all flags passed, the contents of fm_flags will be unmodified.
It is up to userspace to determine whether rejection of a particular
flag is fatal to its operation. This scheme is intended to allow the
fiemap interface to grow in the future but without losing
compatibility with old software.

fm_extent_count specifies the number of elements in the fm_extents[] array
that can be used to return extents. If fm_extent_count is zero, then the
fm_extents[] array is ignored (no extents will be returned), and the
fm_mapped_extents count will hold the number of extents needed in
fm_extents[] to hold the file's current mapping. Note that there is
nothing to prevent the file from changing between calls to FIEMAP.

The following flags can be set in fm_flags:

* FIEMAP_FLAG_SYNC
If this flag is set, the kernel will sync the file before mapping extents.

* FIEMAP_FLAG_XATTR
If this flag is set, the extents returned will describe the inodes
extended attribute lookup tree, instead of its data tree.

Extent Mapping

Extent information is returned within the embedded fm_extents array
which userspace must allocate along with the fiemap structure. The
number of elements in the fiemap_extents[] array should be passed via
fm_extent_count. The number of extents mapped by kernel will be
returned via fm_mapped_extents. If the number of fiemap_extents
allocated is less than would be required to map the requested range,
the maximum number of extents that can be mapped in the fm_extent[]
array will be returned and fm_mapped_extents will be equal to
fm_extent_count. In that case, the last extent in the array will not
complete the requested range and will not have the FIEMAP_EXTENT_LAST
flag set (see the next section on extent flags).

Each extent is described by a single fiemap_extent structure as
returned in fm_extents.

```
struct fiemap_extent {
        __u64   fe_logical;  /* logical offset in bytes for the start of
                              * the extent */
        __u64   fe_physical; /* physical offset in bytes for the start
                              * of the extent */
        __u64   fe_length;   /* length in bytes for the extent */
        __u64   fe_reserved64[2];
        __u32   fe_flags;    /* FIEMAP_EXTENT_* flags for this extent */
        __u32   fe_reserved[3];
};
```

All offsets and lengths are in bytes and mirror those on disk. It is valid
for an extents logical offset to start before the request or its logical
length to extend past the request. Unless FIEMAP_EXTENT_NOT_ALIGNED is
returned, fe_logical, fe_physical, and fe_length will be aligned to the
block size of the file system. With the exception of extents flagged as
FIEMAP_EXTENT_MERGED, adjacent extents will not be merged.

The fe_flags field contains flags which describe the extent returned.
A special flag, FIEMAP_EXTENT_LAST is always set on the last extent in
the file so that the process making fiemap calls can determine when no
more extents are available, without having to call the ioctl again.

Some flags are intentionally vague and will always be set in the
presence of other more specific flags. This way a program looking for
a general property does not have to know all existing and future flags
which imply that property.

For example, if FIEMAP_EXTENT_DATA_INLINE or FIEMAP_EXTENT_DATA_TAIL
are set, FIEMAP_EXTENT_NOT_ALIGNED will also be set. A program looking
for inline or tail-packed data can key on the specific flag. Software
which simply cares not to try operating on non-aligned extents
however, can just key on FIEMAP_EXTENT_NOT_ALIGNED, and not have to

worry about all present and future flags which might imply unaligned
data. Note that the opposite is not true - it would be valid for
FIEMAP_EXTENT_NOT_ALIGNED to appear alone.

* FIEMAP_EXTENT_LAST
This is the last extent in the file. A mapping attempt past this
extent will return nothing.

* FIEMAP_EXTENT_UNKNOWN
The location of this extent is currently unknown. This may indicate
the data is stored on an inaccessible volume or that no storage has
been allocated for the file yet.

* FIEMAP_EXTENT_DELALLOC
  - This will also set FIEMAP_EXTENT_UNKNOWN.
Delayed allocation - while there is data for this extent, its
physical location has not been allocated yet.

* FIEMAP_EXTENT_ENCODED
This extent does not consist of plain filesystem blocks but is
encoded (e.g. encrypted or compressed). Reading the data in this
extent via I/O to the block device will have undefined results.

Note that it is *always* undefined to try to update the data
in-place by writing to the indicated location without the
assistance of the filesystem, or to access the data using the
information returned by the FIEMAP interface while the filesystem
is mounted. In other words, user applications may only read the
extent data via I/O to the block device while the filesystem is
unmounted, and then only if the FIEMAP_EXTENT_ENCODED flag is
clear; user applications must not try reading or writing to the
filesystem via the block device under any other circumstances.

* FIEMAP_EXTENT_DATA_ENCRYPTED
  - This will also set FIEMAP_EXTENT_ENCODED
The data in this extent has been encrypted by the file system.

* FIEMAP_EXTENT_NOT_ALIGNED
Extent offsets and length are not guaranteed to be block aligned.

* FIEMAP_EXTENT_DATA_INLINE
  This will also set FIEMAP_EXTENT_NOT_ALIGNED
Data is located within a meta data block.

* FIEMAP_EXTENT_DATA_TAIL
  This will also set FIEMAP_EXTENT_NOT_ALIGNED
Data is packed into a block with data from other files.

* FIEMAP_EXTENT_UNWRITTEN
Unwritten extent - the extent is allocated but its data has not been
initialized. This indicates the extent's data will be all zero if read
through the filesystem but the contents are undefined if read directly from
the device.

* FIEMAP_EXTENT_MERGED
This will be set when a file does not support extents, i.e., it uses a block

based addressing scheme. Since returning an extent for each block back to
userspace would be highly inefficient, the kernel will try to merge most
adjacent blocks into 'extents'.

VFS -> File System Implementation

File systems wishing to support fiemap must implement a ->fiemap callback on
their inode_operations structure. The fs ->fiemap call is responsible for
defining its set of supported fiemap flags, and calling a helper function on
each discovered extent:

```
struct inode_operations {
       ...

       int (*fiemap)(struct inode *, struct fiemap_extent_info *, u64 start,
                     u64 len);
```

->fiemap is passed struct fiemap_extent_info which describes the
fiemap request:

```
struct fiemap_extent_info {
        unsigned int fi_flags;          /* Flags as passed from user */
        unsigned int fi_extents_mapped; /* Number of mapped extents */
        unsigned int fi_extents_max;    /* Size of fiemap_extent array */
        struct fiemap_extent *fi_extents_start; /* Start of fiemap_extent array
*/
};
```

It is intended that the file system should not need to access any of this
structure directly.

Flag checking should be done at the beginning of the ->fiemap callback via the
fiemap_check_flags() helper:

```
int fiemap_check_flags(struct fiemap_extent_info *fieinfo, u32 fs_flags);
```

The struct fieinfo should be passed in as received from ioctl_fiemap(). The
set of fiemap flags which the fs understands should be passed via fs_flags. If
fiemap_check_flags finds invalid user flags, it will place the bad values in
fieinfo->fi_flags and return -EBADR. If the file system gets -EBADR, from
fiemap_check_flags(), it should immediately exit, returning that error back to
ioctl_fiemap().

For each extent in the request range, the file system should call
the helper function, fiemap_fill_next_extent():

```
int fiemap_fill_next_extent(struct fiemap_extent_info *info, u64 logical,
                            u64 phys, u64 len, u32 flags, u32 dev);
```

fiemap_fill_next_extent() will use the passed values to populate the
next free extent in the fm_extents array. 'General' extent flags will
automatically be set from specific flags on behalf of the calling file

system so that the userspace API is not broken.

fiemap_fill_next_extent() returns 0 on success, and 1 when the
user-supplied fm_extents array is full. If an error is encountered
while copying the extent to user memory, -EFAULT will be returned.