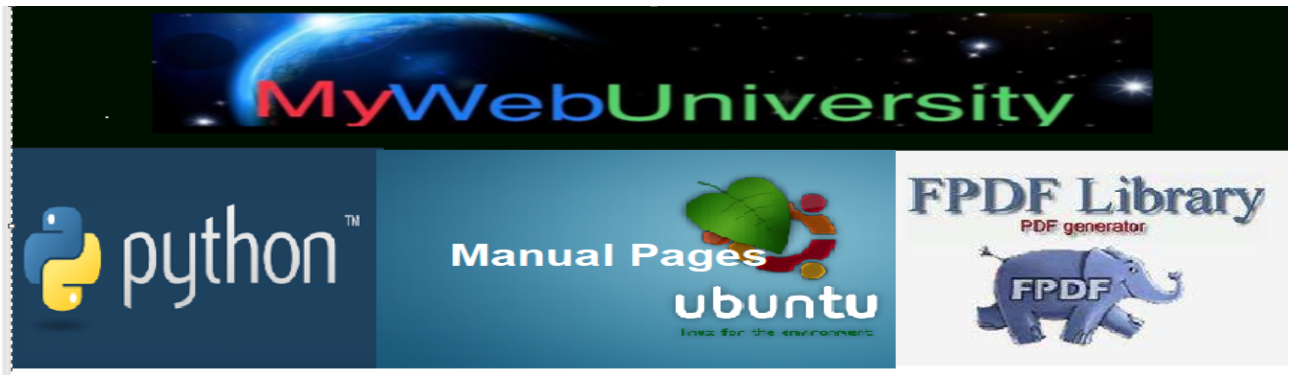

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'ioctl_fsmap.2'

$ man ioctl_fsmap.2

ioctl_fsmap(2) System Calls Manual ioctl_fsmap(2)

NAME

ioctl_fsmap, FS_IOC_GETFSMAP - retrieve the physical layout of the filesystem

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <linux/fsmap.h>  /* Definition of FS_IOC_GETFSMAP,
                             FM?_OF_*, and *FMR_OWN_* constants */
#include <sys/ioctl.h>
int ioctl(int fd, FS_IOC_GETFSMAP, struct fsmap_head * arg);
```

DESCRIPTION

This ioctl(2) operation retrieves physical extent mappings for a filesystem. This informa? tion can be used to discover which files are mapped to a physical block, examine free space, or find known bad blocks, among other things.

The sole argument to this operation should be a pointer to a single struct fsmap_head:

```
struct fsmap {
    __u32 fmr_device;      /* Device ID */
    __u32 fmr_flags;       /* Mapping flags */
    __u64 fmr_physical;    /* Device offset of segment */
    __u64 fmr_owner;       /* Owner ID */
    __u64 fmr_offset;      /* File offset of segment */
    __u64 fmr_length;      /* Length of segment */
    __u64 fmr_reserved[3]; /* Must be zero */
```

```
};
struct fsmap_head {
    __u32 fmh_iflags;       /* Control flags */
    __u32 fmh_oflags;       /* Output flags */
    __u32 fmh_count;        /* # of entries in array including input */
    __u32 fmh_entries;      /* # of entries filled in (output) */
    __u64 fmh_reserved[6];  /* Must be zero */
    struct fsmap fmh_keys[2];  /* Low and high keys for
                                  the mapping search */
    struct fsmap fmh_recs[];   /* Returned records */
};
```

The two fmh_keys array elements specify the lowest and highest reverse-mapping key for which the application would like physical mapping information. A reverse mapping key consists of the tuple (device, block, owner, offset). The owner and offset fields are part of the key because some filesystems support sharing physical blocks between multiple files and there? fore may return multiple mappings for a given physical block.

Filesystem mappings are copied into the fmh_recs array, which immediately follows the header data.

Fields of struct fsmap_head

The fmh_iflags field is a bit mask passed to the kernel to alter the output. No flags are currently defined, so the caller must set this value to zero.

The fmh_oflags field is a bit mask of flags set by the kernel concerning the returned map? pings. If FMH_OF_DEV_T is set, then the fmr_device field represents a dev_t structure con? taining the major and minor numbers of the block device.

The fmh_count field contains the number of elements in the array being passed to the kernel. If this value is 0, fmh_entries will be set to the number of records that would have been returned had the array been large enough; no mapping information will be returned.

The fmh_entries field contains the number of elements in the fmh_recs array that contain useful information.

The fmh_reserved fields must be set to zero.

Keys

The two key records in fsmap_head.fmh_keys specify the lowest and highest extent records in the keyspace that the caller wants returned. A filesystem that can share blocks between

files likely requires the tuple (device, physical, owner, offset, flags) to uniquely index any filesystem mapping record. Classic non-sharing filesystems might be able to identify any record with only (device, physical, flags). For example, if the low key is set to (8:0, 36864, 0, 0, 0), the filesystem will only return records for extents starting at or above 36 KiB on disk. If the high key is set to (8:0, 1048576, 0, 0, 0), only records below 1 MiB will be returned. The format of fmr_device in the keys must match the format of the same field in the output records, as defined below. By convention, the field fsmap_head.fmh_keys[0] must contain the low key and fsmap_head.fmh_keys[1] must contain the high key for the operation.

For convenience, if fmr_length is set in the low key, it will be added to fmr_block or fmr_offset as appropriate. The caller can take advantage of this subtlety to set up subse? quent calls by copying fsmap_head.fmh_recs[fsmap_head.fmh_entries - 1] into the low key. The function fsmap_advance (defined in linux/fsmap.h) provides this functionality.

Fields of struct fsmap

The fmr_device field uniquely identifies the underlying storage device. If the FMH_OF_DEV_T flag is set in the header's fmh_oflags field, this field contains a dev_t from which major and minor numbers can be extracted. If the flag is not set, this field contains a value that must be unique for each unique storage device.

The fmr_physical field contains the disk address of the extent in bytes.

The fmr_owner field contains the owner of the extent. This is an inode number unless FMR_OF_SPECIAL_OWNER is set in the fmr_flags field, in which case the value is determined by the filesystem. See the section below about owner values for more details.

The fmr_offset field contains the logical address in the mapping record in bytes. This field has no meaning if the FMR_OF_SPECIAL_OWNER or FMR_OF_EXTENT_MAP flags are set in fmr_flags.

The fmr_length field contains the length of the extent in bytes.

The fmr_flags field is a bit mask of extent state flags. The bits are:

FMR_OF_PREALLOC

The extent is allocated but not yet written.

FMR_OF_ATTR_FORK

This extent contains extended attribute data.

FMR_OF_EXTENT_MAP

This extent contains extent map information for the owner.

FMR_OF_SHARED

Parts of this extent may be shared.

FMR_OF_SPECIAL_OWNER

The fmr_owner field contains a special value instead of an inode number.

FMR_OF_LAST

This is the last record in the data set.

The fmr_reserved field will be set to zero.

Owner values

Generally, the value of the fmr_owner field for non-metadata extents should be an inode num? ber. However, filesystems are under no obligation to report inode numbers; they may instead report FMR_OWN_UNKNOWN if the inode number cannot easily be retrieved, if the caller lacks sufficient privilege, if the filesystem does not support stable inode numbers, or for any other reason. If a filesystem wishes to condition the reporting of inode numbers based on process capabilities, it is strongly urged that the CAP_SYS_ADMIN capability be used for this purpose.

The following special owner values are generic to all filesystems:

FMR_OWN_FREE

Free space.

FMR_OWN_UNKNOWN

This extent is in use but its owner is not known or not easily retrieved.

FMR_OWN_METADATA

This extent is filesystem metadata.

XFS can return the following special owner values:

XFS_FMR_OWN_FREE

Free space.

XFS_FMR_OWN_UNKNOWN

This extent is in use but its owner is not known or not easily retrieved.

XFS_FMR_OWN_FS

Static filesystem metadata which exists at a fixed address. These are the AG su? perblock, the AGF, the AGFL, and the AGI headers.

XFS_FMR_OWN_LOG

The filesystem journal.

XFS_FMR_OWN_AG

Allocation group metadata, such as the free space btrees and the reverse mapping btrees.

XFS_FMR_OWN_INOBT

The inode and free inode btrees.

XFS_FMR_OWN_INODES

Inode records.

XFS_FMR_OWN_REFC

Reference count information.

XFS_FMR_OWN_COW

This extent is being used to stage a copy-on-write.

XFS_FMR_OWN_DEFECTIVE:

This extent has been marked defective either by the filesystem or the underlying device.

ext4 can return the following special owner values:

EXT4_FMR_OWN_FREE

Free space.

EXT4_FMR_OWN_UNKNOWN

This extent is in use but its owner is not known or not easily retrieved.

EXT4_FMR_OWN_FS

Static filesystem metadata which exists at a fixed address. This is the superblock and the group descriptors.

EXT4_FMR_OWN_LOG

The filesystem journal.

EXT4_FMR_OWN_INODES

Inode records.

EXT4_FMR_OWN_BLKBM

Block bit map.

EXT4_FMR_OWN_INOBM

Inode bit map.

RETURN VALUE

On error, -1 is returned, and errno is set to indicate the error.

ERRORS

The error placed in errno can be one of, but is not limited to, the following:

EBADF fd is not open for reading.

EBADMSG

The filesystem has detected a checksum error in the metadata.

EFAULT The pointer passed in was not mapped to a valid memory address.

EINVAL The array is not long enough, the keys do not point to a valid part of the filesys?

tem, the low key points to a higher point in the filesystem's physical storage ad?

dress space than the high key, or a nonzero value was passed in one of the fields

that must be zero.

ENOMEM Insufficient memory to process the operation.

EOPNOTSUPP

The filesystem does not support this operation.

EUCLEAN

The filesystem metadata is corrupt and needs repair.

STANDARDS

Linux.

Not all filesystems support it.

HISTORY

Linux 4.12.

EXAMPLES

See io/fsmap.c in the xfsprogs distribution for a sample program.

SEE ALSO

ioctl(2)