

**NAME**

`ioctl_fat` – manipulating the FAT filesystem

**SYNOPSIS**

```
#include <linux/msdos_fs.h>
#include <sys/ioctl.h>

int ioctl(int fd, FAT_IOCTL_GET_ATTRIBUTES, uint32_t *attr);
int ioctl(int fd, FAT_IOCTL_SET_ATTRIBUTES, uint32_t *attr);
int ioctl(int fd, FAT_IOCTL_GET_VOLUME_ID, uint32_t *id);
int ioctl(int fd, VFAT_IOCTL_READDIR_BOTH,
          struct __fat_dirent[2] entry);
int ioctl(int fd, VFAT_IOCTL_READDIR_SHORT,
          struct __fat_dirent[2] entry);
```

**DESCRIPTION**

The `ioctl(2)` system call can be used to read and write metadata of FAT filesystems that are not accessible using other system calls.

**Reading and setting file attributes**

Files and directories in the FAT filesystem possess an attribute bit mask that can be read with `FAT_IOCTL_GET_ATTRIBUTES` and written with `FAT_IOCTL_SET_ATTRIBUTES`.

The `fd` argument contains a file descriptor for a file or directory. It is sufficient to create the file descriptor by calling `open(2)` with the `O_RDONLY` flag.

The `attr` argument contains a pointer to a bit mask. The bits of the bit mask are:

**ATTR\_RO**

This bit specifies that the file or directory is read-only.

**ATTR\_HIDDEN**

This bit specifies that the file or directory is hidden.

**ATTR\_SYS**

This bit specifies that the file is a system file.

**ATTR\_VOLUME**

This bit specifies that the file is a volume label. This attribute is read-only.

**ATTR\_DIR**

This bit specifies that this is a directory. This attribute is read-only.

**ATTR\_ARCH**

This bit indicates that this file or directory should be archived. It is set when a file is created or modified. It is reset by an archiving system.

The zero value `ATTR_NONE` can be used to indicate that no attribute bit is set.

**Reading the volume ID**

FAT filesystems are identified by a volume ID. The volume ID can be read with `FAT_IOCTL_GET_VOLUME_ID`.

The `fd` argument can be a file descriptor for any file or directory of the filesystem. It is sufficient to create the file descriptor by calling `open(2)` with the `O_RDONLY` flag.

The `id` argument is a pointer to the field that will be filled with the volume ID. Typically the volume ID is displayed to the user as a group of two 16-bit fields:

```
printf("Volume ID %04x-%04x\n", id >> 16, id & 0xFFFF);
```

**Reading short file names of a directory**

A file or directory on a FAT filesystem always has a short filename consisting of up to 8 capital letters, optionally followed by a period and up to 3 capital letters for the file extension. If the actual filename does not fit into this scheme, it is stored as a long filename of up to 255 UTF-16 characters.

The short filenames in a directory can be read with **VFAT\_IOCTL\_READDIR\_SHORT**. **VFAT\_IOCTL\_READDIR\_BOTH** reads both the short and the long filenames.

The *fd* argument must be a file descriptor for a directory. It is sufficient to create the file descriptor by calling **open(2)** with the **O\_RDONLY** flag. The file descriptor can be used only once to iterate over the directory entries by calling **ioctl(2)** repeatedly.

The *entry* argument is a two-element array of the following structures:

```
struct __fat_dirent {
    long      d_ino;
    __kernel_off_t d_off;
    uint32_t short d_reclen;
    char      d_name[256];
};
```

The first entry in the array is for the short filename. The second entry is for the long filename.

The *d\_ino* and *d\_off* fields are filled only for long filenames. The *d\_ino* field holds the inode number of the directory. The *d\_off* field holds the offset of the file entry in the directory. As these values are not available for short filenames, the user code should simply ignore them.

The field *d\_reclen* contains the length of the filename in the field *d\_name*. To keep backward compatibility, a length of 0 for the short filename signals that the end of the directory has been reached. However, the preferred method for detecting the end of the directory is to test the **ioctl(2)** return value. If no long filename exists, field *d\_reclen* is set to 0 and *d\_name* is a character string of length 0 for the long filename.

## RETURN VALUE

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

For **VFAT\_IOCTL\_READDIR\_BOTH** and **VFAT\_IOCTL\_READDIR\_SHORT** a return value of 1 signals that a new directory entry has been read and a return value of 0 signals that the end of the directory has been reached.

## ERRORS

### ENOENT

This error is returned by **VFAT\_IOCTL\_READDIR\_BOTH** and **VFAT\_IOCTL\_READDIR\_SHORT** if the file descriptor *fd* refers to a removed, but still open directory.

### ENOTDIR

This error is returned by **VFAT\_IOCTL\_READDIR\_BOTH** and **VFAT\_IOCTL\_READDIR\_SHORT** if the file descriptor *fd* does not refer to a directory.

### ENOTTY

The file descriptor *fd* does not refer to an object in a FAT filesystem.

For further error values, see **ioctl(2)**.

## VERSIONS

**VFAT\_IOCTL\_READDIR\_BOTH** and **VFAT\_IOCTL\_READDIR\_SHORT** first appeared in Linux 2.0.

**FAT\_IOCTL\_GET\_ATTRIBUTES** and **FAT\_IOCTL\_SET\_ATTRIBUTES** first appeared in Linux 2.6.12.

**FAT\_IOCTL\_GET\_VOLUME\_ID** was introduced in version 3.11 of the Linux kernel.

## CONFORMING TO

This API is Linux-specific.

## EXAMPLE

### Toggling the archive flag

The following program demonstrates the usage of **ioctl(2)** to manipulate file attributes. The program reads and displays the archive attribute of a file. After inverting the value of the attribute, the program reads and displays the attribute again.

The following was recorded when applying the program for the file */mnt/user/foo*:

```
# ./toggle_fat_archive_flag /mnt/user/foo
Archive flag is set
Toggling archive flag
Archive flag is not set
```

**Program source (toggle\_fat\_archive\_flag.c)**

```
#include <fcntl.h>
#include <linux/msdos_fs.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <unistd.h>

/*
 * Read file attributes of a file on a FAT filesystem.
 * Output the state of the archive flag.
 */
static uint32_t
readattr(int fd)
{
    uint32_t attr;
    int ret;

    ret = ioctl(fd, FAT_IOCTL_GET_ATTRIBUTES, &attr);
    if (ret == -1) {
        perror("ioctl");
        exit(EXIT_FAILURE);
    }

    if (attr & ATTR_ARCH)
        printf("Archive flag is set\n");
    else
        printf("Archive flag is not set\n");

    return attr;
}

int
main(int argc, char *argv[])
{
    uint32_t attr;
    int fd;
    int ret;

    if (argc != 2) {
        printf("Usage: %s FILENAME\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        perror("open");
    }
}
```

```

        exit(EXIT_FAILURE);
    }

    /*
     * Read and display the FAT file attributes.
     */
    attr = readattr(fd);

    /*
     * Invert archive attribute.
     */
    printf("Toggling archive flag\n");
    attr ^= ATTR_ARCH;

    /*
     * Write the changed FAT file attributes.
     */
    ret = ioctl(fd, FAT_IOCTL_SET_ATTRIBUTES, &attr);
    if (ret == -1) {
        perror("ioctl");
        exit(EXIT_FAILURE);
    }

    /*
     * Read and display the FAT file attributes.
     */
    readattr(fd);

    close(fd);

    exit(EXIT_SUCCESS);
}

```

### Reading the volume ID

The following program demonstrates the use of **ioctl(2)** to display the volume ID of a FAT filesystem.

The following output was recorded when applying the program for directory */mnt/user*:

```

$ ./display_fat_volume_id /mnt/user
Volume ID 6443-6241

```

### Program source (display\_fat\_volume\_id.c)

```

#include <fcntl.h>
#include <linux/msdos_fs.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    uint32_t id;
    int fd;
    int ret;

```

```

    if (argc != 2) {
        printf("Usage: %s FILENAME\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    /*
     * Read volume ID.
     */
    ret = ioctl(fd, FAT_IOCTL_GET_VOLUME_ID, &id);
    if (ret == -1) {
        perror("ioctl");
        exit(EXIT_FAILURE);
    }

    /*
     * Format the output as two groups of 16 bits each.
     */
    printf("Volume ID %04x-%04x\n", id >> 16, id & 0xFFFF);

    close(fd);

    exit(EXIT_SUCCESS);
}

```

### Listing a directory

The following program demonstrates the use of **ioctl(2)** to list a directory.

The following was recorded when applying the program to the directory */mnt/user*:

```

$ ./fat_dir /mnt/user
. -> ''
.. -> ''
ALONGF~1.TXT -> 'a long filename.txt'
UPPER.TXT -> ''
LOWER.TXT -> 'lower.txt'

```

### Program source

```

#include <fcntl.h>
#include <linux/msdos_fs.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    struct __fat_dirent entry[2];
    int fd;
    int ret;

```

```

if (argc != 2) {
    printf("Usage: %s DIRECTORY\n", argv[0]);
    exit(EXIT_FAILURE);
}

/*
 * Open file descriptor for the directory.
 */
fd = open(argv[1], O_RDONLY | O_DIRECTORY);
if (fd == -1) {
    perror("open");
    exit(EXIT_FAILURE);
}

for (;;) {

    /*
     * Read next directory entry.
     */
    ret = ioctl( fd, VFAT_IOCTL_READDIR_BOTH, entry);

    /*
     * If an error occurs, the return value is -1.
     * If the end of the directory list has been reached,
     * the return value is 0.
     * For backward compatibility the end of the directory
     * list is also signaled by d_reclen == 0.
     */
    if (ret < 1)
        break;

    /*
     * Write both the short name and the long name.
     */
    printf("%s -> '%s'\n", entry[0].d_name, entry[1].d_name);
}

if (ret == -1) {
    perror("VFAT_IOCTL_READDIR_BOTH");
    exit(EXIT_FAILURE);
}

/*
 * Close the file descriptor.
 */
close(fd);

exit(EXIT_SUCCESS);
}

```

**SEE ALSO****ioctl(2)****COLOPHON**

This page is part of release 5.02 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at

<https://www.kernel.org/doc/man-pages/>.