NAME

mount - mount filesystem

SYNOPSIS

#include <sys/mount.h>

int mount(const char ****source*, const char *****target***,**
const char *****filesystemtype*, **unsigned long** *mountflags*,
const void *****data*);

DESCRIPTION

mount() attaches the filesystem specified by *source* (which is often a device name, but can also be a directory name or a dummy) to the directory specified by *target*.

Appropriate privilege (Linux: the **CAP_SYS_ADMIN** capability) is required to mount filesystems.

Since Linux 2.4 a single filesystem can be visible at multiple mount points, and multiple mounts can be stacked on the same mount point.

Values for the *filesystemtype* argument supported by the kernel are listed in */proc/filesystems* (e.g., minix, ext2, ext3, jfs, xfs, reiserfs, msdos, proc, nfs, iso9660). Further types may become available when the appropriate modules are loaded.

The *mountflags* argument may have the magic number 0xC0ED (**MS_MGC_VAL**) in the top 16 bits (this was required in kernel versions prior to 2.4, but is no longer required and ignored if specified), and various mount flags in the low order 16 bits:

MS_BIND (Linux 2.4 onward)
: Perform a bind mount, making a file or a directory subtree visible at another point within a filesystem. Bind mounts may cross filesystem boundaries and span chroot(2) jails. The *filesystemtype* and *data* arguments are ignored. Up until Linux 2.6.26, *mountflags* was also ignored (the bind mount has the same mount options as the underlying mount point).

MS_DIRSYNC (since Linux 2.5.19)
: Make directory changes on this filesystem synchronous. (This property can be obtained for individual directories or subtrees using chattr(1).)

MS_MANDLOCK
: Permit mandatory locking on files in this filesystem. (Mandatory locking must still be enabled on a per-file basis, as described in fcntl(2).)

MS_MOVE
: Move a subtree. *sour ce* specifies an existing mount point and *target* specifies the new location. The move is atomic: at no point is the subtree unmounted. The*filesystemtyp e*, *mountflags*, and *data* arguments are ignored.

MS_NOATIME
: Do not update access times for (all types of) files on this filesystem.

MS_NODEV
: Do not allow access to devices (special files) on this filesystem.

MS_NODIRATIME
: Do not update access times for directories on this filesystem. This flag provides a subset of the functionality provided by **MS_NOATIME**; that is, **MS_NOATIME** implies **MS_NODIRATIME**.

MS_NOEXEC
: Do not allow programs to be executed from this filesystem.

MS_NOSUID
Do not honor set-user-ID and set-group-ID bits when executing programs from this filesystem.

MS_RDONLY
Mount filesystem read-only.

MS_RELATIME (since Linux 2.6.20)
When a file on this filesystem is accessed, update the file's last access time (atime) only if the current value of atime is less than or equal to the file's last modification time (mtime) or last status change time (ctime). This option is useful for programs, such as **mutt(1),** that need to know when a file has been read since it was last modified. Since Linux 2.6.30, the kernel defaults to the behavior provided by this flag (unless **MS_NOATIME** was specified), and the **MS_STRICTATIME** flag is required to obtain traditional semantics. In addition, since Linux 2.6.30, the file's last access time is always updated if it is more than 1 day old.

MS_REMOUNT
Remount an existing mount. This allows you to change the *mountflags* and *data* of an existing mount without having to unmount and remount the filesystem. *tar get* should be the same value specified in the initial **mount**() call; *source* and *filesystemtype* are ignored. The *mountflags* and *data* arguments should match the values used in the original **mount**() call, except for those parameters that are being deliberately changed.

The following *mountflags* can be changed: **MS_RDONLY**, **MS_SYNCHRONOUS**, **MS_MANDLOCK**; before kernel 2.6.16, the following could also be changed: **MS_NOATIME** and **MS_NODIRATIME**; and, additionally, before kernel 2.4.10, the following could also be changed: **MS_NOSUID**, **MS_NODEV**, **MS_NOEXEC**.

MS_SILENT (since Linux 2.6.17)
Suppress the display of certain (*printk*()) warning messages in the kernel log. This flag supersedes the misnamed and obsolete **MS_VERBOSE** flag (available since Linux 2.4.12), which has the same meaning.

MS_STRICTATIME (since Linux 2.6.30)
Always update the last access time (atime) when files on this filesystem are accessed. (This was the default behavior before Linux 2.6.30.) Specifying this flag overrides the effect of setting the **MS_NOATIME** and **MS_RELATIME** flags.

MS_SYNCHRONOUS
Make writes on this filesystem synchronous (as though the **O_SYNC** flag to open(2) was specified for all file opens to this filesystem).

From Linux 2.4 onward, the **MS_NODEV**, **MS_NOEXEC**, and **MS_NOSUID** flags are settable on a per-mount-point basis. From kernel 2.6.16 onward, **MS_NOATIME** and **MS_NODIRATIME** are also settable on a per-mount-point basis. The **MS_RELATIME** flag is also settable on a per-mount-point basis.

The *data* argument is interpreted by the different filesystems. Typically it is a string of comma-separated options understood by this filesystem. See mount(8) for details of the options available for each filesystem type.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

The error values given below result from filesystem type independent errors. Each filesystem type may have its own special errors and its own special behavior. See the Linux kernel source code for details.

EACCES
A component of a path was not searchable. (See also path_resolution(7).) Or, mounting a read-only filesystem was attempted without giving the **MS_RDONLY** flag. Or, the block device *source* is located on a filesystem mounted with the **MS_NODEV** option.

EBUSY
source is already mounted. Or, it cannot be remounted read-only, because it still holds files open for writing. Or, it cannot be mounted on *target* because *target* is still busy (it is the working directory of some thread, the mount point of another device, has open files, etc.).

EFAULT
One of the pointer arguments points outside the user address space.

EINVAL
source had an invalid superblock. Or, a remount (**MS_REMOUNT**) was attempted, but *source* was not already mounted on *target*. Or, a move (**MS_MOVE**) was attempted, but *source* was not a mount point, or was /.

ELOOP
Too many links encountered during pathname resolution. Or, a move was attempted, while *target* is a descendant of *source*.

EMFILE
(In case no block device is required:) Table of dummy devices is full.

ENAMETOOLONG
A pathname was longer than **MAXPATHLEN**.

ENODEV
filesystemtype not configured in the kernel.

ENOENT
A pathname was empty or had a nonexistent component.

ENOMEM
The kernel could not allocate a free page to copy filenames or data into.

ENOTBLK
source is not a block device (and a device was required).

ENOTDIR
target, or a prefix of *source*, is not a directory.

ENXIO
The major number of the block device *source* is out of range.

EPERM
The caller does not have the required privileges.

VERSIONS

The definitions of **MS_DIRSYNC**, **MS_MOVE**, **MS_REC**, **MS_RELATIME**, and **MS_STRICTATIME** were added to glibc headers in version 2.12.

CONFORMING TO

This function is Linux-specific and should not be used in programs intended to be portable.

NOTES

The original **MS_SYNC** flag was renamed **MS_SYNCHRONOUS** in 1.1.69 when a different **MS_SYNC** was added to *<mman.h>*.

Before Linux 2.4 an attempt to execute a set-user-ID or set-group-ID program on a filesystem mounted with **MS_NOSUID** would fail with **EPERM**. Since Linux 2.4 the set-user-ID and set-group-ID bits are just silently ignored in this case.

Per-process namespaces

Starting with kernel 2.4.19, Linux provides per-process mount namespaces. A mount namespace is the set of filesystem mounts that are visible to a process. Mount-point namespaces can be (and usually are) shared between multiple processes, and changes to the namespace (i.e., mounts and unmounts) by one process are visible to all other processes sharing the same namespace. (The pre-2.4.19 Linux situation can be considered as one in which a single namespace was shared by every process on the system.)

A child process created by fork(2) shares its parent's mount namespace; the mount namespace is preserved across an execve(2).

A process can obtain a private mount namespace if: it was created using the clone(2) **CLONE_NEWNS** flag, in which case its new namespace is initialized to be a *copy* of the namespace of the process that called clone(2); or it calls unshare(2) with the **CLONE_NEWNS** flag, which causes the caller's mount namespace to obtain a private copy of the namespace that it was previously sharing with other processes, so that future mounts and unmounts by the caller are invisible to other processes (except child processes that the caller subsequently creates) and vice versa.

The Linux-specific */proc/PID/mounts* file exposes the list of mount points in the mount namespace of the process with the specified ID; see proc(5) for details.

SEE ALSO

lsblk(1), umount(2), namespaces(7), path_resolution(7), mount(8), umount(8)

COLOPHON

This page is part of release 3.74 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 http://www.kernel.org/doc/man-pages/.