NAME

mount, **nmount**, **unmount** **-** mount or dismount a file system

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <sys/param.h>
#include <sys/mount.h>

int
mount(*const char *type*, *const char *dir*, *int flags*, *void *data*);

int
unmount(*const char *dir*, *int flags*);

#include <sys/uio.h>

int
nmount(*struct iovec *iov*, *u_int niov*, *int flags*);

DESCRIPTION

The **mount**() system call grafts a file system object onto the system file tree at the point *dir*. The argument *data* describes the file system object to be mounted. The argument *type* tells the kernel how to interpret *data* (See *type* below). The contents of the file system become available through the new mount point *dir*. Any files in *dir* at the time of a successful mount are swept under the carpet so to speak, and are unavailable until the file system is unmounted.

The **nmount**() system call behaves similarly to **mount**(), except that the mount options (file system type name, device to mount, mount-point name, etc.) are passed as an array of name-value pairs in the array *iov*, containing *niov* elements. The following options are required by all file systems:

fstype	file system type name (e.g., "procfs")
fspath	mount point pathname (e.g., "/proc")

Depending on the file system type, other options may be recognized or required; for example, most disk-based file systems require a "from" option containing the pathname of a special device in addition to the options listed above.

By default only the super-user may call the **mount**() system call. This restriction can be removed by setting the *vfs.usermount* sysctl(8) variable to a non-zero value; see the BUGS section for more information.

The following *flags* may be specified to suppress default semantics which affect file system access.

MNT_RDONLY	The file system should be treated as read-only; even the super-user may not write on it. Specifying MNT_UPDATE without this option will upgrade a read-only file system to read/write.
MNT_NOEXEC	Do not allow files to be executed from the file system.
MNT_NOSUID	Do not honor setuid or setgid bits on files when executing them. This flag is set automatically when the caller is not the super-user.
MNT_NOATIME	Disable update of file access times.
MNT_SNAPSHOT	Create a snapshot of the file system. This is currently only supported on UFS2 file systems, see mksnap_ffs(8) for more information.
MNT_SUIDDIR	Directories with the SUID bit set chown new files to their own owner. This flag requires the SUIDDIR option to have been compiled into the kernel to have any effect. See the mount(8) and chmod(2) pages for more information.
MNT_SYNCHRONOUS	All I/O to the file system should be done synchronously.
MNT_ASYNC	All I/O to the file system should be done asynchronously.
MNT_FORCE	Force a read-write mount even if the file system appears to be unclean. Dangerous. Together with MNT_UPDATE and MNT_RDONLY, specify that the file system is to be forcibly downgraded to a read-only mount even if some files are open for writing.
MNT_NOCLUSTERR	Disable read clustering.
MNT_NOCLUSTERW	Disable write clustering.
MNT_NOCOVER	Do not mount over the root of another mount point.
MNT_EMPTYDIR	Require an empty directory for the mount point directory.

The flag MNT_UPDATE indicates that the mount command is being applied to an already mounted file system. This allows the mount flags to be changed without requiring that the file system be unmounted and remounted. Some file systems may not allow all flags to be changed. For example, many file systems will not allow a change from read-write to read-only.

The flag MNT_RELOAD causes the vfs subsystem to update its data structures pertaining to the specified already mounted file system.

The *type* argument names the file system. The types of file systems known to the system can be obtained with lsvfs(1).

The *data* argument is a pointer to a structure that contains the type specific arguments to mount. The format for these argument structures is described in the manual page for each file system. By convention file system manual pages are named by prefixing ‘‘mount_’’ to the name of the file system as returned by lsvfs(1). Thus the NFS file system is described by the mount_nfs(8) manual page. It should be noted that a manual page for default file systems, known as UFS and UFS2, does not exist.

The **unmount**() system call disassociates the file system from the specified mount point *dir*.

The *flags* argument may include MNT_FORCE to specify that the file system should be forcibly unmounted even if files are still active. Active special devices continue to work, but any further accesses to any other active files result in errors even if the file system is later remounted.

If the MNT_BYFSID flag is specified, *dir* should instead be a file system ID encoded as "FSID:*val0*:*val1*", where *val0* and *val1* are the contents of the *fsid_t val[]* array in decimal. The file system that has the specified file system ID will be unmounted.

RETURN VALUES

Upon successful completion, the value 0 is returned; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

The **mount**() and **nmount**() system calls will fail when one of the following occurs:

[EPERM]	The caller is neither the super-user nor the owner of *dir*.
[ENAMETOOLONG]	A component of a pathname exceeded 255 characters, or the entire length of a path name exceeded 1023 characters.

[ELOOP]	Too many symbolic links were encountered in translating a pathname.
[ENOENT]	A component of *dir* does not exist.
[ENOTDIR]	A component of *name* is not a directory, or a path prefix of *special* is not a directory.
[EBUSY]	Another process currently holds a reference to *dir*.
[EBUSY]	The MNT_NOCOVER option was given, and the requested mount point is already the root of another mount point.
[EFAULT]	The *dir* argument points outside the process's allocated address space.
[EIO]	An I/O error occurred while reading data from *special*.
[EINTEGRITY]	The backing store for *special* detected corrupted data while reading.

The following errors can occur for a *ufs* file system mount:

[ENODEV]	A component of ufs_args *fspec* does not exist.
[ENOTBLK]	The *fspec* argument is not a block device.
[ENOTEMPTY]	The MNT_EMPTYDIR option was specified, and the requested mount point is not an empty directory.
[ENXIO]	The major device number of *fspec* is out of range (this indicates no device driver exists for the associated hardware).
[EBUSY]	*fspec* is already mounted.
[EMFILE]	No space remains in the mount table.
[EINVAL]	The super block for the file system had a bad magic number or an out of range block size.
[EINTEGRITY]	The super block for the file system had a bad check hash. The check hash can usually be corrected by running fsck(8).

[ENOMEM]	Not enough memory was available to read the cylinder group information for the file system.
[EIO]	An I/O error occurred while reading the super block or cylinder group information.
[EFAULT]	The *fspec* argument points outside the process's allocated address space.

The following errors can occur for a *nfs* file system mount:

[ETIMEDOUT]	*Nfs* timed out trying to contact the server.
[EFAULT]	Some part of the information described by nfs_args points outside the process's allocated address space.

The **unmount**() system call may fail with one of the following errors:

[EPERM]	The caller is neither the super-user nor the user who issued the corresponding **mount**() call.
[ENAMETOOLONG]	The length of the path name exceeded 1023 characters.
[EINVAL]	The requested directory is not in the mount table.
[ENOENT]	The file system ID specified using MNT_BYFSID was not found in the mount table.
[EINVAL]	The file system ID specified using MNT_BYFSID could not be decoded.
[EINVAL]	The specified file system is the root file system.
[EBUSY]	A process is holding a reference to a file located on the file system.
[EIO]	An I/O error occurred while writing cached file system information.
[EFAULT]	The *dir* argument points outside the process's allocated address space.

SEE ALSO

lsvfs(1), mksnap_ffs(8), mount(8), umount(8)

HISTORY

The **mount**() and **unmount**() functions appeared in Version 1 AT&T UNIX. The **nmount**() system call first appeared in FreeBSD 5.0.

BUGS

Some of the error codes need translation to more obvious messages.

Allowing untrusted users to mount arbitrary media, e.g. by enabling *vfs.usermount*, should not be considered safe. Most file systems in FreeBSD were not built to safeguard against malicious devices.