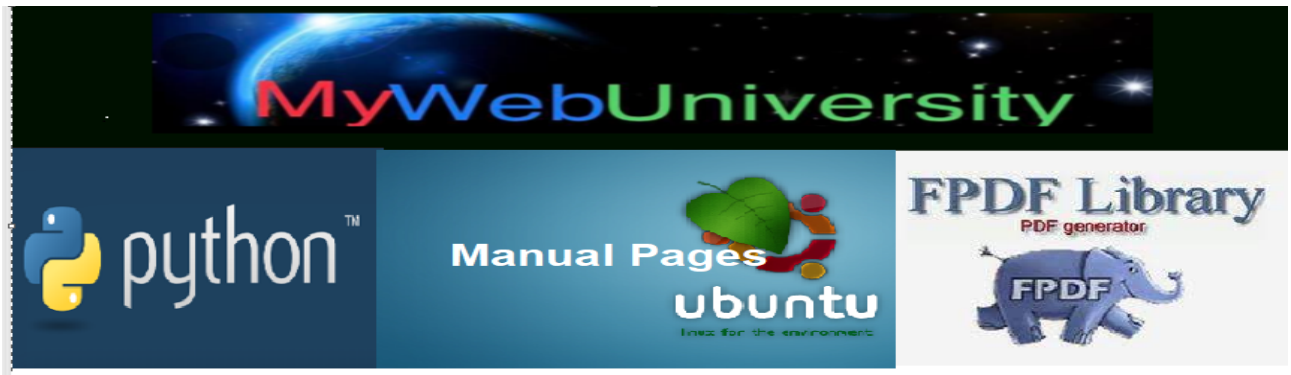

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man fsmount.2

fsmount(2) System Calls Manual fsmount(2)

NAME

fsmount - instantiate mount object from filesystem context

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

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

int fsmount(int fsfd, unsigned int flags, unsigned int attr_flags);
```

DESCRIPTION

The fsmount() system call is part of the suite of file-descriptor-based mount facilities in Linux.

fsmount() creates a new detached mount object for the root of the new filesystem instance referenced by the filesystem context file descriptor fsfd. A new file descriptor associated with the detached mount object is then returned. In order to create a mount object with fs? mount(), the calling process must have the CAP_SYS_ADMIN capability.

The filesystem context must have been created with a call to fsopen(2) and then had a filesystem instance instantiated with a call to fsconfig(2) with FSCONFIG_CMD_CREATE or FSCONFIG_CMD_CREATE_EXCL in order to be in the correct state for this operation (the "await? ing-mount" mode in kernel-developer parlance). Unlike open_tree(2) with OPEN_TREE_CLONE, fsmount() can only be called once in the lifetime of a filesystem context to produce a mount object.

As with file descriptors returned from open_tree(2) called with OPEN_TREE_CLONE, the re? turned file descriptor can then be used with move_mount(2), mount_setattr(2), or other such

system calls to do further mount operations. This mount object will be unmounted and de? stroyed when the file descriptor is closed if it was not otherwise attached to a mount point by calling move_mount(2). (Note that the unmount operation on close(2) is lazy?akin to calling umount2(2) with MNT_DETACH; any existing open references to files from the mount ob? ject will continue to work, and the mount object will only be completely destroyed once it ceases to be busy.) The returned file descriptor also acts the same as one produced by open(2) with O_PATH, meaning it can also be used as a dirfd argument to "*at()" system calls.

flags controls the creation of the returned file descriptor. A value for flags is con? structed by bitwise ORing zero or more of the following constants:

FSMOUNT_CLOEXEC

Set the close-on-exec (FD_CLOEXEC) flag on the new file descriptor. See the description of the O_CLOEXEC flag in open(2) for reasons why this may be use? ful.

attr_flags specifies mount attributes which will be applied to the created mount object, in the form of MOUNT_ATTR_* flags. The flags are interpreted as though mount_setattr(2) was called with attr.attr_set set to the same value as attr_flags. MOUNT_ATTR_* flags which would require specifying additional fields in mount_attr(2type) (such as MOUNT_ATTR_IDMAP) are not valid flag values for attr_flags.

If the fsmount() operation is successful, the filesystem context associated with the file descriptor fsfd is reset and placed into reconfiguration mode, as if it were just returned by fspick(2). You may continue to use fsconfig(2) with the now-reset filesystem context, including issuing the FSCONFIG_CMD_RECONFIGURE command to reconfigure the filesystem in? stance.

RETURN VALUE

On success, a new file descriptor is returned. On error, -1 is returned, and errno is set to indicate the error.

ERRORS

EBUSY The filesystem context associated with fsfd is not in the right state to be used by fsmount().

EINVAL flags had an invalid flag set.

EINVAL attr_flags had an invalid MOUNT_ATTR_* flag set.

EMFILE The calling process has too many open files to create more.

ENFILE The system has too many open files to create more.

ENOSPC The "anonymous" mount namespace necessary to contain the new mount object could not be allocated, as doing so would exceed the configured per-user limit on the number of mount namespaces in the current user namespace. (See also namespaces(7).)

ENOMEM The kernel could not allocate sufficient memory to complete the operation.

EPERM The calling process does not have the required CAP_SYS_ADMIN capability.

STANDARDS

Linux.

HISTORY

Linux 5.2. glibc 2.36.

EXAMPLES

```
int fsfd, mntfd, tmpfd;
fsfd = fsopen("tmpfs", FSOPEN_CLOEXEC);
fsconfig(fsfd, FSCONFIG_CMD_CREATE, NULL, NULL, 0);
mntfd = fsmount(fsfd, FSMOUNT_CLOEXEC,
                MOUNT_ATTR_NODEV | MOUNT_ATTR_NOEXEC);
/* Create a new file without attaching the mount object */
tmpfd = openat(mntfd, "tmpfile", O_CREAT | O_EXCL | O_RDWR, 0600);
unlinkat(mntfd, "tmpfile", 0);
/* Attach the mount object to "/tmp" */
move_mount(mntfd, "", AT_FDCWD, "/tmp", MOVE_MOUNT_F_EMPTY_PATH);
```

SEE ALSO

fsconfig(2), fsopen(2), fspick(2), mount(2), mount_setattr(2), move_mount(2), open_tree(2), mount_namespaces(7)