NAME

rename − change the name or location of a file

SYNOPSIS

#include <stdio.h>

int rename(const char ****oldpath*, const char ****newpath***);**

DESCRIPTION

rename() renames a file, moving it between directories if required. Any other hard links to the file (as created using **link**(2)) are unaffected. Open file descriptors for *oldpath* are also unaffected.

If *newpath* already exists it will be atomically replaced (subject to a few conditions; see ERRORS below), so that there is no point at which another process attempting to access *newpath* will find it missing.

If *oldpath* and *newpath* are existing hard links referring to the same file, then **rename**() does nothing, and returns a success status.

If *newpath* exists but the operation fails for some reason **rename**() guarantees to leave an instance of *newpath* in place.

oldpath can specify a directory. In this case, *newpath* must either not exist, or it must specify an empty directory.

However, when overwriting there will probably be a window in which both *oldpath* and *newpath* refer to the file being renamed.

If *oldpath* refers to a symbolic link the link is renamed; if *newpath* refers to a symbolic link the link will be overwritten.

RETURN VALUE

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

ERRORS

EACCES

Write permission is denied for the directory containing *oldpath* or *newpath*, or, search permission is denied for one of the directories in the path prefix of *oldpath* or *newpath*, or *oldpath* is a directory and does not allow write permission (needed to update the .. entry). (See also **path_resolution**(7).)

EBUSY

The rename fails because *oldpath* or *newpath* is a directory that is in use by some process (perhaps as current working directory, or as root directory, or because it was open for reading) or is in use by the system (for example as mount point), while the system considers this an error. (Note that there is no requirement to return **EBUSY** in such cases — there is nothing wrong with doing the rename anyway — but it is allowed to return **EBUSY** if the system cannot otherwise handle such situations.)

EFAULT

oldpath or *newpath* points outside your accessible address space.

EINVAL

The new pathname contained a path prefix of the old, or, more generally, an attempt was made to make a directory a subdirectory of itself.

EISDIR

newpath is an existing directory, but *oldpath* is not a directory.

ELOOP
Too many symbolic links were encountered in resolving *oldpath* or *newpath*.

EMLINK
oldpath already has the maximum number of links to it, or it was a directory and the directory containing *newpath* has the maximum number of links.

ENAMETOOLONG
oldpath or *newpath* was too long.

ENOENT
The link named by *oldpath* does not exist; or, a directory component in *newpath* does not exist; or, *oldpath* or *newpath* is an empty string.

ENOMEM
Insufficient kernel memory was available.

ENOSPC
The device containing the file has no room for the new directory entry.

ENOTDIR
A component used as a directory in *oldpath* or *newpath* is not, in fact, a directory. Or, *oldpath* is a directory, and *newpath* exists but is not a directory.

ENOTEMPTY or **EEXIST**
newpath is a non-empty directory, that is, contains entries other than "." and "..".

EPERM or **EACCES**
The directory containing *oldpath* has the sticky bit (**S_ISVTX**) set and the process's effective user ID is neither the user ID of the file to be deleted nor that of the directory containing it, and the process is not privileged (Linux: does not have the **CAP_FOWNER** capability); or *newpath* is an existing file and the directory containing it has the sticky bit set and the process's effective user ID is neither the user ID of the file to be replaced nor that of the directory containing it, and the process is not privileged (Linux: does not have the **CAP_FOWNER** capability); or the file system containing *pathname* does not support renaming of the type requested.

EROFS
The file is on a read-only file system.

EXDEV
oldpath and *newpath* are not on the same mounted file system. (Linux permits a file system to be mounted at multiple points, but **rename**() does not work across different mount points, even if the same file system is mounted on both.)

CONFORMING TO
4.3BSD, C89, C99, POSIX.1-2001.

BUGS
On NFS file systems, you can not assume that if the operation failed the file was not renamed. If the server does the rename operation and then crashes, the retransmitted RPC which will be processed when the server is up again causes a failure. The application is expected to deal with this. See **link**(2) for a similar problem.

SEE ALSO
mv(1), **chmod**(2), **link**(2), **renameat**(2), **symlink**(2), **unlink**(2), **path_resolution**(7), **symlink**(7)

COLOPHON
This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.