NAME

link, **linkat** - make a hard file link

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <unistd.h>

int
link(*const char *name1*, *const char *name2*);

int
linkat(*int fd1*, *const char *name1*, *int fd2*, *const char *name2*, *int flag*);

DESCRIPTION

The **link**() system call atomically creates the specified directory entry (hard link) *name2* with the attributes of the underlying object pointed at by *name1*. If the link is successful: the link count of the underlying object is incremented; *name1* and *name2* share equal access and rights to the underlying object.

If *name1* is removed, the file *name2* is not deleted and the link count of the underlying object is decremented.

The object pointed at by the *name1* argument must exist for the hard link to succeed and both *name1* and *name2* must be in the same file system. The *name1* argument may not be a directory.

The **linkat**() system call is equivalent to *link* except in the case where either *name1* or *name2* or both are relative paths. In this case a relative path *name1* is interpreted relative to the directory associated with the file descriptor *fd1* instead of the current working directory and similarly for *name2* and the file descriptor *fd2*.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in <*fcntl.h*>:

AT_SYMLINK_FOLLOW
: If *name1* names a symbolic link, a new link for the target of the symbolic link is created.

AT_RESOLVE_BENEATH
: Only walk paths below the directory specified by the *fd1* descriptor. See the description of the

O_RESOLVE_BENEATH flag in the open(2) manual page.

AT_EMPTY_PATH

If the *name1* argument is an empty string, link the file referenced by the descriptor *fd1*. The operation requires that the calling process has the PRIV_VFS_FHOPEN privilege, effectively being executed with effective user root.

If **linkat**() is passed the special value AT_FDCWD in the *fd1* or *fd2* parameter, the current working directory is used for the respective *name* argument. If both *fd1* and *fd2* have value AT_FDCWD, the behavior is identical to a call to **link**(). Unless *flag* contains the AT_SYMLINK_FOLLOW flag, if *name1* names a symbolic link, a new link is created for the symbolic link *name1* and not its target.

RETURN VALUES

The **link**() function returns the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

The **link**() system call will fail and no link will be created if:

[ENOTDIR]	A component of either path prefix is not a directory.
[ENAMETOOLONG]	A component of either pathname exceeded 255 characters, or entire length of either path name exceeded 1023 characters.
[ENOENT]	A component of either path prefix does not exist.
[EOPNOTSUPP]	The file system containing the file named by *name1* does not support links.
[EMLINK]	The link count of the file named by *name1* would exceed 32767.
[EACCES]	A component of either path prefix denies search permission.
[EACCES]	The requested link requires writing in a directory with a mode that denies write permission.
[ELOOP]	Too many symbolic links were encountered in translating one of the pathnames.
[ENOENT]	The file named by *name1* does not exist.

[EEXIST]	The link named by *name2* does exist.
[EPERM]	The file named by *name1* is a directory.
[EPERM]	The file named by *name1* has its immutable or append-only flag set, see the chflags(2) manual page for more information.
[EPERM]	The parent directory of the file named by *name2* has its immutable flag set.
[EXDEV]	The link named by *name2* and the file named by *name1* are on different file systems.
[ENOSPC]	The directory in which the entry for the new link is being placed cannot be extended because there is no space left on the file system containing the directory.
[EDQUOT]	The directory in which the entry for the new link is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.
[EIO]	An I/O error occurred while reading from or writing to the file system to make the directory entry.
[EINTEGRITY]	Corrupted data was detected while reading from the file system.
[EROFS]	The requested link requires writing in a directory on a read-only file system.
[EFAULT]	One of the pathnames specified is outside the process's allocated address space.

In addition to the errors returned by the **link**(), the **linkat**() system call may fail if:

[EBADF]	The *name1* or *name2* argument does not specify an absolute path and the *fd1* or *fd2* argument, respectively, is neither AT_FDCWD nor a valid file descriptor open for searching.
[EINVAL]	The value of the *flag* argument is not valid.
[ENOTDIR]	The *name1* or *name2* argument is not an absolute path and *fd1* or *fd2*, respectively, is neither AT_FDCWD nor a file descriptor associated with a directory.

[ENOTCAPABLE]	*name1* is not strictly relative to the starting directory. For example, *name1* is absolute or includes a ".." component that escapes the directory hierarchy specified by *fd*, and the process is in capability mode or the AT_RESOLVE_BENEATH flag was specified.

SEE ALSO

chflags(2), readlink(2), symlink(2), unlink(2)

STANDARDS

The **link**() system call is expected to conform to IEEE Std 1003.1-1990 ("POSIX.1"). The **linkat**() system call follows The Open Group Extended API Set 2 specification.

HISTORY

The **link**() function appeared in Version 1 AT&T UNIX. The **linkat**() system call appeared in FreeBSD 8.0.

The **link**() system call traditionally allows the super-user to link directories which corrupts the file system coherency. This implementation no longer permits it.