

**NAME**

mknod, mknodat – create a special or ordinary file

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int mknod(const char *pathname, mode_t mode, dev_t dev);

#include <fcntl.h>      /* Definition of AT_* constants */
#include <sys/stat.h>

int mknodat(int dirfd, const char *pathname, mode_t mode, dev_t dev);
```

Feature Test Macro Requirements for glibc (see [feature\\_test\\_macros\(7\)](#)):

```
mknod():
    _XOPEN_SOURCE >= 500 || /* Since glibc 2.19: */ _DEFAULT_SOURCE || /* Glibc versions <=
    2.19: */ _BSD_SOURCE || _SVID_SOURCE
```

**DESCRIPTION**

The system call **mknod()** creates a filesystem node (file, device special file, or named pipe) named *pathname*, with attributes specified by *mode* and *dev*.

The *mode* argument specifies both the file mode to use and the type of node to be created. It should be a combination (using bitwise OR) of one of the file types listed below and zero or more of the file mode bits listed in [inode\(7\)](#).

The file mode is modified by the process's *umask* in the usual way: in the absence of a default ACL, the permissions of the created node are (*mode* &  $\sim$ *umask*).

The file type must be one of **S\_IFREG**, **S\_IFCHR**, **S\_IFBLK**, **S\_IFIFO**, or **S\_IFSOCK** to specify a regular file (which will be created empty), character special file, block special file, FIFO (named pipe), or UNIX domain socket, respectively. (Zero file type is equivalent to type **S\_IFREG**.)

If the file type is **S\_IFCHR** or **S\_IFBLK**, then *dev* specifies the major and minor numbers of the newly created device special file ([makedev\(3\)](#) may be useful to build the value for *dev*); otherwise it is ignored.

If *pathname* already exists, or is a symbolic link, this call fails with an **EEXIST** error.

The newly created node will be owned by the effective user ID of the process. If the directory containing the node has the set-group-ID bit set, or if the filesystem is mounted with BSD group semantics, the new node will inherit the group ownership from its parent directory; otherwise it will be owned by the effective group ID of the process.

**mknodat()**

The **mknodat()** system call operates in exactly the same way as **mknod()**, except for the differences described here.

If the *pathname* given in *pathname* is relative, then it is interpreted relative to the directory referred to by the file descriptor *dirfd* (rather than relative to the current working directory of the calling process, as is done by **mknod()** for a relative *pathname*).

If *pathname* is relative and *dirfd* is the special value **AT\_FDCWD**, then *pathname* is interpreted relative to the current working directory of the calling process (like **mknod()**).

If *pathname* is absolute, then *dirfd* is ignored.

See [openat\(2\)](#) for an explanation of the need for **mknodat()**.

**RETURN VALUE**

**mknod()** and **mknodat()** return zero on success, or  $-1$  if an error occurred (in which case, *errno* is set appropriately).

## ERRORS

### EACCES

The parent directory does not allow write permission to the process, or one of the directories in the path prefix of *pathname* did not allow search permission. (See also [path\\_resolution\(7\)](#).)

### EDQUOT

The user's quota of disk blocks or inodes on the filesystem has been exhausted.

### EEXIST

*pathname* already exists. This includes the case where *pathname* is a symbolic link, dangling or not.

### EFAULT

*pathname* points outside your accessible address space.

### EINVAL

*mode* requested creation of something other than a regular file, device special file, FIFO or socket.

### ELOOP

Too many symbolic links were encountered in resolving *pathname*.

### ENAMETOOLONG

*pathname* was too long.

### ENOENT

A directory component in *pathname* does not exist or is a dangling symbolic link.

### ENOMEM

Insufficient kernel memory was available.

### ENOSPC

The device containing *pathname* has no room for the new node.

### ENOTDIR

A component used as a directory in *pathname* is not, in fact, a directory.

### EPERM

*mode* requested creation of something other than a regular file, FIFO (named pipe), or UNIX domain socket, and the caller is not privileged (Linux: does not have the **CAP\_MKNOD** capability); also returned if the filesystem containing *pathname* does not support the type of node requested.

### EROFS

*pathname* refers to a file on a read-only filesystem.

The following additional errors can occur for **mknodat()**:

### EBADF

*dirfd* is not a valid file descriptor.

### ENOTDIR

*pathname* is relative and *dirfd* is a file descriptor referring to a file other than a directory.

## VERSIONS

**mknodat()** was added to Linux in kernel 2.6.16; library support was added to glibc in version 2.4.

## CONFORMING TO

**mknod()**: SVr4, 4.4BSD, POSIX.1-2001 (but see below), POSIX.1-2008.

**mknodat()**: POSIX.1-2008.

## NOTES

POSIX.1-2001 says: "The only portable use of **mknod()** is to create a FIFO-special file. If *mode* is not **S\_IFIFO** or *dev* is not 0, the behavior of **mknod()** is unspecified." However, nowadays one should never use **mknod()** for this purpose; one should use [mkfifo\(3\)](#), a function especially defined for this purpose.

Under Linux, **mknod()** cannot be used to create directories. One should make directories with [mkdir\(2\)](#).

There are many infelicities in the protocol underlying NFS. Some of these affect **mknod()** and **mknodat()**.

**SEE ALSO**

[mknod\(1\)](#), [chmod\(2\)](#), [chown\(2\)](#), [fcntl\(2\)](#), [mkdir\(2\)](#), [mount\(2\)](#), [socket\(2\)](#), [stat\(2\)](#), [umask\(2\)](#), [unlink\(2\)](#), [makedev\(3\)](#), [mkfifo\(3\)](#), [acl\(5\)](#) [path\\_resolution\(7\)](#)

**COLOPHON**

This page is part of release 4.16 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 <https://www.kernel.org/doc/man-pages/>.