NAME

mknod – make a directory, or a special or ordinary file

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>

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

DESCRIPTION

mknod( ) creates a new file named by the path name pointed to by *path*. The file type and permissions of the new file are initialized from *mode*.

The file type is specified in *mode* by the **S_IFMT** bits, which must be set to one of the following values:

S_IFIFO	fifo special
S_IFCHR	character special
S_IFDIR	directory
S_IFBLK	block special
S_IFREG	ordinary file

The file access permissions are specified in *mode* by the 0007777 bits, and may be constructed by an OR of the following values:

S_ISUID	04000	Set user ID on execution.
S_ISGID	020#0	Set group ID on execution if # is **7**, **5**, **3**, or **1**. Enable mandatory file/record locking if # is **6**, **4**, **2**, or **0**.
S_ISVTX	01000	Save text image after execution.
S_IRWXU	00700	Read, write, execute by owner.
S_IRUSR	00400	Read by owner.
S_IWUSR	00200	Write by owner.
S_IXUSR	00100	Execute (search if a directory) by owner.
S_IRWXG	00070	Read, write, execute by group.
S_IRGRP	00040	Read by group.
S_IWGRP	00020	Write by group.
S_IXGRP	00010	Execute by group.
S_IRWXO	00007	Read, write, execute (search) by others.
S_IROTH	00004	Read by others.
S_IWOTH	00002	Write by others
S_IXOTH	00001	Execute by others.

The owner ID of the file is set to the effective user ID of the process. The group ID of the file is set to the effective group ID of the process. However, if the **S_ISGID** bit is set in the parent directory, then the group ID of the file is inherited from the parent. If the group ID of the new file does not match the effective group ID or one of the supplementary group IDs, the **S_ISGID** bit is cleared.

The access permission bits of *mode* are modified by the process's file mode creation mask: all bits set in the process's file mode creation mask are cleared (see **umask**(2)). If *mode* indicates a block or character special file, *dev* is a configuration-dependent specification of a character or block I/O device. If *mode* does not indicate a block special or character special device, *dev* is ignored. See **makedev**(3C).

mknod( ) may be invoked only by a privileged user for file types other than FIFO special.

If *path* is a symbolic link, it is not followed.

RETURN VALUES

0 is returned upon successful completion; otherwise, –**1** is returned and **errno** is set to indicate the error.

ERRORS

mknod( ) fails and creates no new file if one or more of the following are true:

EEXIST The named file exists.

EFAULT *path* points to an illegal address.

EINTR A signal was caught during the **mknod()** function.

EINVAL *dev* is invalid.

ELOOP Too many symbolic links were encountered in translating *path*.

EMULTIHOP Components of *path* require hopping to multiple remote machines and the file system type does not allow it.

ENAMETOOLONG The length of the *path* argument exceeds {**PATH_MAX**}, or the length of a *path* component exceeds {**NAME_MAX**} while {**_POSIX_NO_TRUNC**} is in effect.

ENOENT A component of the path prefix does not exist or is a null path-name.

ENOLINK *path* points to a remote machine and the link to that machine is no longer active.

ENOSPC No space is available.

ENOTDIR A component of the path prefix is not a directory.

EPERM The effective user of the calling process is not super-user.

EROFS The directory in which the file is to be created is located on a read-only file system.

SEE ALSO

chmod(2), **exec**(2), **mkdir**(2), **umask**(2), **makedev**(3C), **mkfifo**(3C), **stat**(5)

NOTES

Normally, applications should use the **mkdir**(2) routine to make a directory, since the function **mknod( )** may not establish directory entries for **.** (the directory itself) and **..** (the parent directory), and appropriate permissions are not required. Similarly, **mkfifo**(3C) should be used in preference to **mknod( )** in order to create FIFOs.