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

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'mknod.3p' command

$ man mknod.3p

MKNOD(3P) POSIX Programmer's Manual MKNOD(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mknod, mknodat ? make directory, special file, or regular file

SYNOPSIS

```
#include <sys/stat.h>
int mknod(const char *path, mode_t mode, dev_t dev);
#include <fcntl.h>
int mknodat(int fd, const char *path, mode_t mode, dev_t dev);
```

DESCRIPTION

The mknod() function shall create a new file named by the pathname to which the argument path points.

The file type for path is OR'ed into the mode argument, and the appli? cation shall select one of the following symbolic constants:

```
??????????????????????????????????????????????
? Name ? Description ?
??????????????????????????????????????????????
?S_IFIFO ? FIFO-special ?
?S_IFCHR ? Character-special (non-portable) ?
```

?S_IFDIR ? Directory (non-portable) ?

?S_IFBLK ? Block-special (non-portable) ?

?S_IFREG ? Regular (non-portable) ?

??

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 unspec? ified.

The permissions for the new file are OR'ed into the mode argument, and may be selected from any combination of the following symbolic con? stants:

??

? Name ? Description ?

??

?S_ISUID ? Set user ID on execution. ?

?S_ISGID ? Set group ID on execution. ?

?S_IRWXU ? Read, write, or execute (search) by owner. ?

?S_IRUSR ? Read by owner. ?

?S_IWUSR ? Write by owner. ?

?S_IXUSR ? Execute (search) by owner. ?

?S_IRWXG ? Read, write, or execute (search) by group. ?

?S_IRGRP ? Read by group. ?

?S_IWGRP ? Write by group. ?

?S_IXGRP ? Execute (search) by group. ?

?S_IRWXO ? Read, write, or execute (search) by others. ?

?S_IROTH ? Read by others. ?

?S_IWOTH ? Write by others. ?

?S_IXOTH ? Execute (search) by others. ?

?S_ISVTX ? On directories, restricted deletion flag. ?

??

The user ID of the file shall be initialized to the effective user ID of the process. The group ID of the file shall be initialized to either the effective group ID of the process or the group ID of the parent di? rectory. Implementations shall provide a way to initialize the file's

group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process. The owner, group, and other permission bits of mode shall be modified by the file mode creation mask of the process. The mknod() function shall clear each bit whose corresponding bit in the file mode creation mask of the process is set.

If path names a symbolic link, mknod() shall fail and set errno to [EEXIST].

Upon successful completion, mknod() shall mark for update the last data access, last data modification, and last file status change timestamps of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

Only a process with appropriate privileges may invoke mknod() for file types other than FIFO-special.

The mknodat() function shall be equivalent to the mknod() function ex? cept in the case where path specifies a relative path. In this case the newly created directory, special file, or regular file is located rela? tive to the directory associated with the file descriptor fd instead of the current working directory. If the access mode of the open file de? scription associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If mknodat() is passed the special value AT_FDCWD in the fd parameter, the current working directory shall be used and the behavior shall be identical to a call to mknod().

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set errno to indicate the error. If -1 is returned, the new file shall not be created.

ERRORS

These functions shall fail if:

EACCES A component of the path prefix denies search permission, or write permission is denied on the parent directory.

EEXIST The named file exists.

EINVAL An invalid argument exists.

EIO An I/O error occurred while accessing the file system.

ELOOP A loop exists in symbolic links encountered during resolution of the path argument.

ENAMETOOLONG

The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT A component of the path prefix of path does not name an existing file or path is an empty string.

ENOENT or ENOTDIR

The path argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If path without the trailing <slash> characters would name an existing file, an [ENOENT] error shall not occur.

ENOSPC The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.

ENOTDIR

A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.

EPERM The invoking process does not have appropriate privileges and the file type is not FIFO-special.

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

The mknodat() function shall fail if:

EACCES The access mode of the open file description associated with fd is not O_SEARCH and the permissions of the directory underlying fd do not permit directory searches.

EBADF The path argument does not specify an absolute path and the fd argument is neither AT_FDCWD nor a valid file descriptor open

for reading or searching.

ENOTDIR

The path argument is not an absolute path and fd is a file de?

scriptor associated with a non-directory file.

These functions may fail if:

ELOOP More than {SYMLOOP_MAX} symbolic links were encountered during

resolution of the path argument.

ENAMETOOLONG

The length of a pathname exceeds {PATH_MAX}, or pathname resolu?

tion of a symbolic link produced an intermediate result with a

length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Creating a FIFO Special File

The following example shows how to create a FIFO special file named /home/cnd/mod_done, with read/write permissions for owner, and with read permissions for group and others.

```
#include <sys/types.h>
#include <sys/stat.h>
dev_t dev;
int   status;
...
status  = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
    S_IRUSR | S_IRGRP | S_IROTH, dev);
```

APPLICATION USAGE

The mkfifo() function is preferred over this function for making FIFO special files.

RATIONALE

The POSIX.1?1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151?2 required that implementa? tions provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also sup?

porting a way to set the group ID to the effective group ID of the cre?
ating process. Conforming applications should not assume which group
ID will be used. If it matters, an application can use chown() to set
the group ID after the file is created, or determine under what condi?
tions the implementation will set the desired group ID.

The purpose of the mknodat() function is to create directories, special
files, or regular files in directories other than the current working
directory without exposure to race conditions. Any part of the path of
a file could be changed in parallel to a call to mknod(), resulting in
unspecified behavior. By opening a file descriptor for the target di?
rectory and using the mknodat() function it can be guaranteed that the
newly created directory, special file, or regular file is located rela?
tive to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), creat(), exec, fstatat(), mkdir(), mkfifo(), open(), umask()

The Base Definitions volume of POSIX.1?2017, <fcntl.h>, <sys_stat.h>

COPYRIGHT