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 'mkfifo.3p' command

$ man mkfifo.3p

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

mkfifo, mkfifoat ? make a FIFO special file

SYNOPSIS

```
#include <sys/stat.h>
int mkfifo(const char *path, mode_t mode);
#include <fcntl.h>
int mkfifoat(int fd, const char *path, mode_t mode);
```

DESCRIPTION

The mkfifo() function shall create a new FIFO special file named by the pathname pointed to by path. The file permission bits of the new FIFO shall be initialized from mode. The file permission bits of the mode argument shall be modified by the process' file creation mask.

When bits in mode other than the file permission bits are set, the ef? fect is implementation-defined.

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

The FIFO's user ID shall be set to the process' effective user ID. The

FIFO's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementa? tion-defined way to initialize the FIFO's group ID to the effective group ID of the calling process.

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

The mkfifoat() function shall be equivalent to the mkfifo() function except in the case where path specifies a relative path. In this case the newly created FIFO is created relative to the directory associated with the file descriptor fd instead of the current working directory.

If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether di? rectory searches are permitted using the current permissions of the di? rectory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If mkfifoat() 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 mkfifo().

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, no FIFO shall 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 of the FIFO to be created.

EEXIST The named file already exists.

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.

EROFS The named file resides on a read-only file system.

The mkfifoat() 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 File

The following example shows how to create a FIFO 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>
int status;
...
status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
    S_IRGRP | S_IROTH);
```

APPLICATION USAGE

None.

RATIONALE

The syntax of this function is intended to maintain compatibility with historical implementations of mknod(). The latter function was in? cluded in the 1984 /usr/group standard but only for use in creating FIFO special files. The mknod() function was originally excluded from the POSIX.1?1988 standard as implementation-defined and replaced by mkdir() and mkfifo(). The mknod() function is now included for align? ment with the Single UNIX Specification.

The POSIX.1?1990 standard required that the group ID of a newly created FIFO 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 FIFO is created, or determine under what condi?

tions the implementation will set the desired group ID.

The purpose of the mkfifoat() function is to create a FIFO special file in directories other than the current working directory without expo? sure to race conditions. Any part of the path of a file could be changed in parallel to a call to mkfifo(), resulting in unspecified be? havior. By opening a file descriptor for the target directory and using the mkfifoat() function it can be guaranteed that the newly created FIFO is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), mknod(), umask()

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

COPYRIGHT