fdopendir - Man Page

open directory associated with file descriptor

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.

Synopsis

```
#include <dirent.h>

DIR *fdopendir(int fd);
DIR *opendir(const char *dirname);
```

Description

The *fdopendir*() function shall be equivalent to the *opendir*() function except that the directory is specified by a file descriptor rather than by a name. The file offset associated with the file descriptor at the time of the call determines which entries are returned.

Upon successful return from *fdopendir*(), the file descriptor is under the control of the system, and if any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of *closedir*(), *readdir*(), *readdir_r*(), *rewinddir*(), or *seekdir*(), the behavior is undefined. Upon calling *closedir*() the file descriptor shall be closed.

fdopendir - Man Page

The *opendir*() function shall open a directory stream corresponding to the directory named by the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is implemented using a file descriptor, applications shall only be able to open up to a total of {OPEN_MAX} files and directories.

If the type **DIR** is implemented using a file descriptor, the descriptor shall be obtained as if the O_DIRECTORY flag was passed to *open*().

Return Value

Upon successful completion, these functions shall return a pointer to an object of type **DIR**. Otherwise, these functions shall return a null pointer and set *errno* to indicate the error.

Errors

The *fdopendir*() function shall fail if:

EBADF

The *fd* argument is not a valid file descriptor open for reading.

ENOTDIR

The descriptor *fd* is not associated with a directory.

The *opendir*() function shall fail if:

EACCES

Search permission is denied for the component of the path prefix of *dirname* or read permission is denied for *dirname*.

ELOOP

fdopendir - Man Page

ENAMETOOLONG

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

ENOENT

A component of *dirname* does not name an existing directory or *dirname* is an empty string.

ENOTDIR

A component of *dirname* names an existing file that is neither a directory nor a symbolic link to a directory.

The *opendir*() function may fail if:

ELOOP

More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *dirname* argument.

EMFILE

All file descriptors available to the process are currently open.

ENAMETOOLONG

The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENFILE

Too many files are currently open in the system.

The following sections are informative.

fdopendir - Man Page

-

Open a Directory Stream

The following program fragment demonstrates how the *opendir*() function is used.

```
#include <dirent.h>
...
    DIR *dir;
    struct dirent *dp;
...
    if ((dir = opendir (".")) == NULL) {
        perror ("Cannot open .");
        exit (1);
    }

    while ((dp = readdir (dir)) != NULL) {
...
```

Find And Open a File

The following program searches through a given directory looking for files whose name does not begin with a dot and whose size is larger than 1 MiB.

```
#include <stdio.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[])
```

fdopendir - Man Page

```
        struct dirent *dp;
        int dfd, ffd;

        if ((d = fdopendir((dfd = open("./tmp", O_RDONLY))))
            fprintf(stderr, "Cannot open ./tmp directory\n")
            exit(1);
        }
        while ((dp = readdir(d)) != NULL) {
            if (dp->d_name[0] == '.')
                continue;
            /* there is a possible race condition here as th
             * could be renamed between the readdir and the
            if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) ==
                perror(dp->d_name);
                continue;
            }
            if (fstat(ffd, &statbuf) == 0 && statbuf.st_size
                /* found it ... */
                printf("%s: %jdK\n", dp->d_name,
                    (intmax_t)(statbuf.st_size / 1024));
            }
            close(ffd);
        }
        closedir(d); // note this implicitly closes dfd
        return 0;
    }
```

Application Usage

The *opendir*() function should be used in conjunction with *readdir*(), *closedir*(), and *rewinddir*() to examine the contents of the directory (see the Examples section in *readdir*()). This method is recommended for portability.

fdopendir - Man Page

The purpose of the *fdopendir*() function is to enable opening 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 *opendir*(), resulting in unspecified behavior.

Based on historical implementations, the rules about file descriptors apply to directory streams as well. However, this volume of POSIX.1-2017 does not mandate that the directory stream be implemented using file descriptors. The description of *closedir*() clarifies that if a file descriptor is used for the directory stream, it is mandatory that *closedir*() deallocate the file descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the FD_CLOEXEC had been set for the file descriptor.

The directory entries for dot and dot-dot are optional. This volume of POSIX.1-2017 does not provide a way to test *a priori* for their existence because an application that is portable must be written to look for (and usually ignore) those entries. Writing code that presumes that they are the first two entries does not always work, as many implementations permit them to be other than the first two entries, with a “normal” entry preceding them. There is negligible value in providing a way to determine what the implementation does because the code to deal with dot and dot-dot must be written in any case and because such a flag would add to the list of those flags (which has proven in itself to be objectionable) and might be abused.

Since the structure and buffer allocation, if any, for directory operations are defined by the implementation, this volume of POSIX.1-2017 imposes no portability requirements for erroneous program constructs, erroneous data, or the use of unspecified values such as the use or referencing of a *dirp* value or a **dirent** structure value

fdopendir - Man Page

Future Directions

None.

See Also

closedir(), dirfd(), fstatat(), open(), readdir(), rewinddir(), symlink()

The Base Definitions volume of POSIX.1-2017, <dirent.h>, <sys_types.h>

Copyright

Referenced By

fdopendir - Man Page

symlink(3p), telldir(3p).

2017 IEEE/The Open Group POSIX Programmer's Manual