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 'open.3p' command

$ man open.3p

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

open, openat ? open file

SYNOPSIS

```
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path, int oflag, ...);
int openat(int fd, const char *path, int oflag, ...);
```

DESCRIPTION

The open() function shall establish the connection between a file and a file descriptor. It shall create an open file description that refers to a file and a file descriptor that refers to that open file descrip? tion. The file descriptor is used by other I/O functions to refer to that file. The path argument points to a pathname naming the file.

The open() function shall return a file descriptor for the named file, allocated as described in Section 2.14, File Descriptor Allocation.

The open file description is new, and therefore the file descriptor shall not share it with any other process in the system. The FD_CLOEXEC

file descriptor flag associated with the new file descriptor shall be cleared unless the O_CLOEXEC flag is set in oflag.

The file offset used to mark the current position within the file shall be set to the beginning of the file.

The file status flags and file access modes of the open file descrip? tion shall be set according to the value of oflag.

Values for oflag are constructed by a bitwise-inclusive OR of flags from the following list, defined in <fcntl.h>. Applications shall specify exactly one of the first five values (file access modes) below in the value of oflag:

O_EXEC Open for execute only (non-directory files). The result is unspecified if this flag is applied to a directory.

O_RDONLY Open for reading only.

O_RDWR Open for reading and writing. The result is undefined if this flag is applied to a FIFO.

O_SEARCH Open directory for search only. The result is unspecified if this flag is applied to a non-directory file.

O_WRONLY Open for writing only.

Any combination of the following may be used:

O_APPEND If set, the file offset shall be set to the end of the file prior to each write.

O_CLOEXEC If set, the FD_CLOEXEC flag for the new file descriptor shall be set.

O_CREAT If the file exists, this flag has no effect except as noted under O_EXCL below. Otherwise, if O_DIRECTORY is not set the file shall be created as a regular file; the user ID of the file shall be set to the effective user ID of the process; the group ID of the file shall be set to the group ID of the file's parent directory or to the ef? fective group ID of the process; and the access permis? sion bits (see <sys/stat.h>) of the file mode shall be set to the value of the argument following the oflag ar? gument taken as type mode_t modified as follows: a bit?

wise AND is performed on the file-mode bits and the cor?
responding bits in the complement of the process' file
mode creation mask. Thus, all bits in the file mode whose
corresponding bit in the file mode creation mask is set
are cleared. When bits other than the file permission
bits are set, the effect is unspecified. The argument
following the oflag argument does not affect whether the
file is open for reading, writing, or for both. Implemen?
tations shall provide a way to initialize the file's
group ID to the group ID of the parent directory. Imple?
mentations may, but need not, provide an implementation-
defined way to initialize the file's group ID to the ef?
fective group ID of the calling process.

O_DIRECTORY If path resolves to a non-directory file, fail and set
errno to [ENOTDIR].

O_DSYNC Write I/O operations on the file descriptor shall com?
plete as defined by synchronized I/O data integrity com?
pletion.

O_EXCL If O_CREAT and O_EXCL are set, open() shall fail if the
file exists. The check for the existence of the file and
the creation of the file if it does not exist shall be
atomic with respect to other threads executing open()
naming the same filename in the same directory with
O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set,
and path names a symbolic link, open() shall fail and set
errno to [EEXIST], regardless of the contents of the sym?
bolic link. If O_EXCL is set and O_CREAT is not set, the
result is undefined.

O_NOCTTY If set and path identifies a terminal device, open()
shall not cause the terminal device to become the con?
trolling terminal for the process. If path does not iden?
tify a terminal device, O_NOCTTY shall be ignored.

O_NOFOLLOW If path names a symbolic link, fail and set errno to

[ELOOP].

O_NONBLOCK When opening a FIFO with O_RDONLY or O_WRONLY set:

* If O_NONBLOCK is set, an open() for reading-only shall return without delay. An open() for writing-only shall return an error if no process currently has the file open for reading.
* If O_NONBLOCK is clear, an open() for reading-only shall block the calling thread until a thread opens the file for writing. An open() for writing-only shall block the calling thread until a thread opens the file for reading.

When opening a block special or character special file that supports non-blocking opens:

* If O_NONBLOCK is set, the open() function shall re? turn without blocking for the device to be ready or available. Subsequent behavior of the device is de? vice-specific.
* If O_NONBLOCK is clear, the open() function shall block the calling thread until the device is ready or available before returning.

Otherwise, the O_NONBLOCK flag shall not cause an error, but it is unspecified whether the file status flags will include the O_NONBLOCK flag.

O_RSYNC Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in oflag, all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integ? rity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

O_SYNC Write I/O operations on the file descriptor shall com?

plete as defined by synchronized I/O file integrity com?
pletion.
The O_SYNC flag shall be supported for regular files,
even if the Synchronized Input and Output option is not
supported.

O_TRUNC If the file exists and is a regular file, and the file is
successfully opened O_RDWR or O_WRONLY, its length shall
be truncated to 0, and the mode and owner shall be un?
changed. It shall have no effect on FIFO special files or
terminal device files. Its effect on other file types is
implementation-defined. The result of using O_TRUNC with?
out either O_RDWR or O_WRONLY is undefined.

O_TTY_INIT If path identifies a terminal device other than a pseudo-
terminal, the device is not already open in any process,
and either O_TTY_INIT is set in oflag or O_TTY_INIT has
the value zero, open() shall set any non-standard termios
structure terminal parameters to a state that provides
conforming behavior; see the Base Definitions volume of
POSIX.1?2017, Section 11.2, Parameters that Can be Set.
It is unspecified whether O_TTY_INIT has any effect if
the device is already open in any process. If path iden?
tifies the slave side of a pseudo-terminal that is not
already open in any process, open() shall set any non-
standard termios structure terminal parameters to a state
that provides conforming behavior, regardless of whether
O_TTY_INIT is set. If path does not identify a terminal
device, O_TTY_INIT shall be ignored.

If O_CREAT and O_DIRECTORY are set and the requested access mode is neither O_WRONLY nor O_RDWR, the result is unspecified.

If O_CREAT is set and the file did not previously exist, upon success?ful completion, open() shall mark for update the last data access, last data modification, and last file status change timestamps of the file and the last data modification and last file status change timestamps

of the parent directory.

If O_TRUNC is set and the file did previously exist, upon successful completion, open() shall mark for update the last data modification and last file status change timestamps of the file.

If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.

If path refers to a STREAMS file, oflag may be constructed from O_NON? BLOCK OR'ed with either O_RDONLY, O_WRONLY, or O_RDWR. Other flag val? ues are not applicable to STREAMS devices and shall have no effect on them. The value O_NONBLOCK affects the operation of STREAMS drivers and certain functions applied to file descriptors associated with STREAMS files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.

The application shall ensure that it specifies the O_TTY_INIT flag on the first open of a terminal device since system boot or since the de? vice was closed by the process that last had it open. The application need not specify the O_TTY_INIT flag when opening pseudo-terminals. If path names the master side of a pseudo-terminal device, then it is un? specified whether open() locks the slave side so that it cannot be opened. Conforming applications shall call unlockpt() before opening the slave side.

The largest value that can be represented correctly in an object of type off_t shall be established as the offset maximum in the open file description.

The openat() function shall be equivalent to the open() function except in the case where path specifies a relative path. In this case the file to be opened is determined 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 de? scriptor 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.

The oflag parameter and the optional fourth parameter correspond ex?
actly to the parameters of open().

If openat() 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 open().

RETURN VALUE

Upon successful completion, these functions shall open the file and re?
turn a non-negative integer representing the file descriptor. Other?
wise, these functions shall return -1 and set errno to indicate the er?
ror. If -1 is returned, no files shall be created or modified.

ERRORS

These functions shall fail if:

EACCES Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by oflag are denied, or the file does not exist and write permission is de?nied for the parent directory of the file to be created, or O_TRUNC is specified and write permission is denied.

EEXIST O_CREAT and O_EXCL are set, and the named file exists.

EINTR A signal was caught during open().

EINVAL The implementation does not support synchronized I/O for this file.

EIO The path argument names a STREAMS file and a hangup or error oc?curred during the open().

EISDIR The named file is a directory and oflag includes O_WRONLY or O_RDWR, or includes O_CREAT without O_DIRECTORY.

ELOOP A loop exists in symbolic links encountered during resolution of the path argument, or O_NOFOLLOW was specified and the path ar?gument names a symbolic link.

EMFILE All file descriptors available to the process are currently open.

ENAMETOOLONG

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

ENFILE The maximum allowable number of files is currently open in the system.

ENOENT O_CREAT is not set and a component of path does not name an existing file, or O_CREAT is set and a component of the path prefix of path does not name an existing file, or path points to an empty string.

ENOENT or ENOTDIR

O_CREAT is set, and 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.

ENOSR The path argument names a STREAMS-based file and the system is unable to allocate a STREAM.

ENOSPC The directory or file system that would contain the new file cannot be expanded, the file does not exist, and O_CREAT is specified.

ENOTDIR

A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory; or O_CREAT and O_EXCL are not specified, the path argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters, and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory; or O_DIRECTORY was specified and the path argument resolves to a non-directory file.

ENXIO O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.

ENXIO The named file is a character special or block special file, and the device associated with this special file does not exist.

EOVERFLOW

The named file is a regular file and the size of the file cannot be represented correctly in an object of type off_t.

EROFS The named file resides on a read-only file system and either

O_WRONLY, O_RDWR, O_CREAT (if the file does not exist), or

O_TRUNC is set in the oflag argument.

The openat() 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:

EAGAIN The path argument names the slave side of a pseudo-terminal de?

vice that is locked.

EINVAL The value of the oflag argument is not valid.

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}.

ENOMEM The path argument names a STREAMS file and the system is unable

to allocate resources.

EOPNOTSUPP

The path argument names a socket.

ETXTBSY

The file is a pure procedure (shared text) file that is being

executed and oflag is O_WRONLY or O_RDWR.

The following sections are informative.

EXAMPLES

Opening a File for Writing by the Owner

The following example opens the file /tmp/file, either by creating it

(if it does not already exist), or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file, the access permission bits in the file mode of the file are set to permit reading and writing by the owner, and to permit reading only by group members and others.

If the call to open() is successful, the file is opened for writing.

```
#include <fcntl.h>
...
int fd;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
char *pathname = "/tmp/file";
...
fd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);
...
```

Opening a File Using an Existence Check

The following example uses the open() function to try to create the LOCKFILE file and open it for writing. Since the open() function speci? fies the O_EXCL flag, the call fails if the file already exists. In that case, the program assumes that someone else is updating the pass? word file and exits.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#define LOCKFILE "/etc/ptmp"
...
int pfd; /* Integer for file descriptor returned by open() call. */
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
```

```
...
```

Opening a File for Writing

The following example opens a file for writing, creating the file if it does not already exist. If the file does exist, the system truncates the file to zero bytes.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#define LOCKFILE "/etc/ptmp"
...
int pfd;
char pathname[PATH_MAX+1];
...
if ((pfd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    perror("Cannot open output file\n"); exit(1);
}
...
```

APPLICATION USAGE

POSIX.1?2008 does not require that terminal parameters be automatically set to any state on first open, nor that they be reset after the last close. It is possible for a non-conforming application to leave a ter? minal device in a state where the next process to use that device finds it in a non-conforming state, but has no way of determining this. To ensure that the device is set to a conforming initial state, applica? tions which perform a first open of a terminal (other than a pseudo-terminal) should do so using the O_TTY_INIT flag to set the parameters associated with the terminal to a conforming state.

Except as specified in this volume of POSIX.1?2017, the flags allowed in oflag are not mutually-exclusive and any number of them may be used simultaneously. Not all combinations of flags make sense. For example, using O_SEARCH | O_CREAT will successfully open a pre-existing direc?

tory for searching, but if there is no existing file by that name, then it is unspecified whether a regular file will be created. Likewise, if a non-directory file descriptor is successfully returned, it is unspec? ified whether that descriptor will have execute permissions as if by O_EXEC (note that it is unspecified whether O_EXEC and O_SEARCH have the same value).

RATIONALE

Some implementations permit opening FIFOs with O_RDWR. Since FIFOs could be implemented in other ways, and since two file descriptors can be used to the same effect, this possibility is left as undefined.

See getgroups() about the group of a newly created file.

The use of open() to create a regular file is preferable to the use of creat(), because the latter is redundant and included only for histori? cal reasons.

The use of the O_TRUNC flag on FIFOs and directories (pipes cannot be open()-ed) must be permissible without unexpected side-effects (for ex? ample, creat() on a FIFO must not remove data). Since terminal special files might have type-ahead data stored in the buffer, O_TRUNC should not affect their content, particularly if a program that normally opens a regular file should open the current controlling terminal instead.

Other file types, particularly implementation-defined ones, are left implementation-defined.

POSIX.1?2008 permits [EACCES] to be returned for conditions other than those explicitly listed.

The O_NOCTTY flag was added to allow applications to avoid unintention? ally acquiring a controlling terminal as a side-effect of opening a terminal file. This volume of POSIX.1?2017 does not specify how a con? trolling terminal is acquired, but it allows an implementation to pro? vide this on open() if the O_NOCTTY flag is not set and other condi? tions specified in the Base Definitions volume of POSIX.1?2017, Chapter 11, General Terminal Interface are met.

In historical implementations the value of O_RDONLY is zero. Because of that, it is not possible to detect the presence of O_RDONLY and another

option. Future implementations should encode O_RDONLY and O_WRONLY as bit flags so that:

O_RDONLY | O_WRONLY == O_RDWR

O_EXEC and O_SEARCH are specified as two of the five file access modes. Since O_EXEC does not apply to directories, and O_SEARCH only applies to directories, their values need not be distinct. Since O_RDONLY has historically had the value zero, implementations are not able to dis? tinguish between O_SEARCH and O_SEARCH | O_RDONLY, and similarly for O_EXEC.

In general, the open() function follows the symbolic link if path names a symbolic link. However, the open() function, when called with O_CREAT and O_EXCL, is required to fail with [EEXIST] if path names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This behavior is required so that privileged applications can create a new file in a known location without the possibility that a symbolic link might cause the file to be created in a different location. For example, a privileged application that must create a file with a predictable name in a user-writable directory, such as the user's home directory, could be compromised if the user creates a symbolic link with that name that refers to a nonexistent file in a system directory. If the user can influence the contents of a file, the user could com? promise the system by creating a new system configuration or spool file that would then be interpreted by the system. The test for a symbolic link which refers to a nonexisting file must be atomic with the cre? ation of a new file.

In addition, the open() function refuses to open non-directories if the O_DIRECTORY flag is set. This avoids race conditions whereby a user might compromise the system by substituting a hard link to a sensitive file (e.g., a device or a FIFO) while a privileged application is run? ning, where opening a file even for read access might have undesirable side-effects.

In addition, the open() function does not follow symbolic links if the O_NOFOLLOW flag is set. This avoids race conditions whereby a user

might compromise the system by substituting a symbolic link to a sensi?
tive file (e.g., a device) while a privileged application is running,
where opening a file even for read access might have undesirable side-
effects.

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 openat() function is to enable opening files in di?
rectories 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 open(), resulting in unspecified behavior. By
opening a file descriptor for the target directory and using the ope?
nat() function it can be guaranteed that the opened file is located
relative to the desired directory. Some implementations use the ope?
nat() function for other purposes as well. In some cases, if the oflag
parameter has the O_XATTR bit set, the returned file descriptor pro?
vides access to extended attributes. This functionality is not stan?
dardized here.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), close(), creat(), dirfd(), dup(), exec, fcntl(), fdopendir(),
link(), lseek(), mkdtemp(), mknod(), read(), symlink(), umask(), un?
lockpt(), write()

The Base Definitions volume of POSIX.1?2017, Chapter 11, General Termi?
nal Interface, <fcntl.h>, <sys_stat.h>, <sys_types.h>