## NAME

**open**, **openat** **-** open or create a file for reading, writing or executing

## LIBRARY

Standard C Library (libc, -lc)

## SYNOPSIS

**#include <fcntl.h>**

*int*
**open**(*const char *path*, *int flags*, *...*);

*int*
**openat**(*int fd*, *const char *path*, *int flags*, *...*);

## DESCRIPTION

The file name specified by *path* is opened for either execution or reading and/or writing as specified by the argument *flags* and the file descriptor returned to the calling process. The *flags* argument may indicate the file is to be created if it does not exist (by specifying the O_CREAT flag). In this case **open**() and **openat**() require an additional argument *mode_t mode*, and the file is created with mode *mode* as described in chmod(2) and modified by the process’ umask value (see umask(2)).

The **openat**() function is equivalent to the **open**() function except in the case where the *path* specifies a relative path. For **openat**() and relative *path*, the file to be opened is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. The *flag* parameter and the optional fourth parameter correspond exactly to the parameters of **open**(). If **openat**() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory is used and the behavior is identical to a call to **open**().

When **openat**() is called with an absolute *path*, it ignores the *fd* argument.

In capsicum(4) capability mode, **open**() is not permitted. The *path* argument to **openat**() must be strictly relative to a file descriptor *fd*. *path* must not be an absolute path and must not contain ".." components which cause the path resolution to escape the directory hierarchy starting at *fd*. Additionally, no symbolic link in *path* may target absolute path or contain escaping ".." components. *fd* must not be AT_FDCWD.

If the vfs.lookup_cap_dotdot sysctl(3) MIB is set to zero, ".." components in the paths, used in capability mode, are completely disabled. If the vfs.lookup_cap_dotdot_nonlocal MIB is set to zero, ".." is not allowed if found on non-local filesystem.

The flags specified are formed by *or*’ing the following values

| | |
|---|---|
| O_RDONLY | open for reading only |
| O_WRONLY | open for writing only |
| O_RDWR | open for reading and writing |
| O_EXEC | open for execute only |
| O_SEARCH | open for search only, an alias for O_EXEC |
| O_NONBLOCK | do not block on open |
| O_APPEND | append on each write |
| O_CREAT | create file if it does not exist |
| O_TRUNC | truncate size to 0 |
| O_EXCL | error if create and file exists |
| O_SHLOCK | atomically obtain a shared lock |
| O_EXLOCK | atomically obtain an exclusive lock |
| O_DIRECT | eliminate or reduce cache effects |
| O_FSYNC | synchronous writes (historical synonym for O_SYNC) |
| O_SYNC | synchronous writes |
| O_DSYNC | synchronous data writes |
| O_NOFOLLOW | do not follow symlinks |
| O_NOCTTY | ignored |
| O_TTY_INIT | ignored |
| O_DIRECTORY | error if file is not a directory |
| O_CLOEXEC | set FD_CLOEXEC upon open |
| O_VERIFY | verify the contents of the file |
| O_RESOLVE_BENEATH | path resolution must not cross the fd directory |
| O_PATH | record only the target path in the opened descriptor |
| O_EMPTY_PATH | openat, open file referenced by fd if path is empty |

Opening a file with O_APPEND set causes each write on the file to be appended to the end. If O_TRUNC is specified and the file exists, the file is truncated to zero length. If O_EXCL is set with O_CREAT and the file already exists, **open**() returns an error. This may be used to implement a simple exclusive access locking mechanism. If O_EXCL is set and the last component of the pathname is a symbolic link, **open**() will fail even if the symbolic link points to a non-existent name. If the O_NONBLOCK flag is specified and the **open**() system call would result in the process being blocked for some reason (e.g., waiting for carrier on a dialup line), **open**() returns immediately. The descriptor remains in non-blocking mode for subsequent operations.

If O_SYNC is used in the mask, all writes will immediately and synchronously be written to disk. O_FSYNC is an historical synonym for O_SYNC.

If O_DSYNC is used in the mask, all data and metadata required to read the data will be synchronously written to disk, but changes to metadata such as file access and modification timestamps may be written later.

If O_NOFOLLOW is used in the mask and the target file passed to **open**() is a symbolic link then the **open**() will fail.

When opening a file, a lock with flock(2) semantics can be obtained by setting O_SHLOCK for a shared lock, or O_EXLOCK for an exclusive lock. If creating a file with O_CREAT, the request for the lock will never fail (provided that the underlying file system supports locking).

O_DIRECT may be used to minimize or eliminate the cache effects of reading and writing. The system will attempt to avoid caching the data you read or write. If it cannot avoid caching the data, it will minimize the impact the data has on the cache. Use of this flag can drastically reduce performance if not used with care.

O_NOCTTY may be used to ensure the OS does not assign this file as the controlling terminal when it opens a tty device. This is the default on FreeBSD, but is present for POSIX compatibility. The **open**() system call will not assign controlling terminals on FreeBSD.

O_TTY_INIT may be used to ensure the OS restores the terminal attributes when initially opening a TTY. This is the default on FreeBSD, but is present for POSIX compatibility. The initial call to **open**() on a TTY will always restore default terminal attributes on FreeBSD.

O_DIRECTORY may be used to ensure the resulting file descriptor refers to a directory. This flag can be used to prevent applications with elevated privileges from opening files which are even unsafe to open with O_RDONLY, such as device nodes.

O_CLOEXEC may be used to set FD_CLOEXEC flag for the newly returned file descriptor.

O_VERIFY may be used to indicate to the kernel that the contents of the file should be verified before allowing the open to proceed. The details of what "verified" means is implementation specific. The run-time linker (rtld) uses this flag to ensure shared objects have been verified before operating on them.

O_RESOLVE_BENEATH returns ENOTCAPABLE if any intermediate component of the specified relative path does not reside in the directory hierarchy beneath the starting directory. Absolute paths or even the temporal escape from beneath of the starting directory is not allowed.

When *fd* is opened with O_SEARCH, execute permissions are checked at open time. The *fd* may not be used for any read operations like getdirentries(2). The primary use for this descriptor will be as the lookup descriptor for the ***at**() family of functions.

O_PATH returns a file descriptor that can be used as a directory file descriptor for openat(2) and other system calls taking a file descriptor argument, like fstatat(2) and others. The other functionality of the returned file descriptor is limited to the descriptor-level operations. It can be used for

| | |
|---|---|
| fcntl(2) | but advisory locking is not allowed |
| dup(2) | |
| close(2) | |
| fstat(2) | |
| fexecve(2) | |
| SCM_RIGHTS | can be passed over a unix(4) socket using a SCM_RIGHTS message |
| kqueue(2) | using for EVFILT_VNODE |
| readlinkat(2) | |
| __acl_get_fd(2), __acl_aclcheck_fd(2) | |

But operations like read(2), ftruncate(2), and any other that operate on file and not on file descriptor (except fstat(2) ), are not allowed.

A file descriptor created with the O_PATH flag can be opened into normal (operable) file descriptor by specifying it as the *fd* argument to **openat**() with empty *path* and flag O_EMPTY_PATH. Such an open behaves as if the current path of the file referenced by *fd* is passed, except that the path walk permissions are not checked. See also the description of AT_EMPTY_PATH flag for fstatat(2) and related syscalls.

If successful, **open**() returns a non-negative integer, termed a file descriptor. It returns -1 on failure. The file pointer used to mark the current position within the file is set to the beginning of the file.

If a sleeping open of a device node from devfs(5) is interrupted by a signal, the call always fails with EINTR, even if the SA_RESTART flag is set for the signal. A sleeping open of a fifo (see mkfifo(2)) is restarted as normal.

When a new file is created it is given the group of the directory which contains it.

Unless O_CLOEXEC flag was specified, the new descriptor is set to remain open across execve(2) system calls; see close(2), fcntl(2) and O_CLOEXEC description.

The system imposes a limit on the number of file descriptors open simultaneously by one process. The getdtablesize(2) system call returns the current system limit.

## RETURN VALUES

If successful, **open**() and **openat**() return a non-negative integer, termed a file descriptor. They return -1 on failure, and set *errno* to indicate the error.

## ERRORS

The named file is opened unless:

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

[ENAMETOOLONG]
A component of a pathname exceeded 255 characters, or an entire path name exceeded 1023 characters.

[ENOENT] O_CREAT is not set and the named file does not exist.

[ENOENT] A component of the path name that must exist does not exist.

[EACCES] Search permission is denied for a component of the path prefix.

[EACCES] The required permissions (for reading and/or writing) are denied for the given flags.

[EACCES] O_TRUNC is specified and write permission is denied.

[EACCES] O_CREAT is specified, the file does not exist, and the directory in which it is to be created does not permit writing.

[EPERM] O_CREAT is specified, the file does not exist, and the directory in which it is to be created has its immutable flag set, see the chflags(2) manual page for more information.

[EPERM] The named file has its immutable flag set and the file is to be modified.

[EPERM] The named file has its append-only flag set, the file is to be modified, and O_TRUNC is specified or O_APPEND is not specified.

[ELOOP] Too many symbolic links were encountered in translating the pathname.

[EISDIR] The named file is a directory, and the arguments specify it is to be modified.

[EISDIR] The named file is a directory, and the flags specified O_CREAT without O_DIRECTORY.

[EROFS] The named file resides on a read-only file system, and the file is to be modified.

[EROFS] O_CREAT is specified and the named file would reside on a read-only file system.

[EMFILE] The process has already reached its limit for open file descriptors.

[ENFILE] The system file table is full.

[EMLINK] O_NOFOLLOW was specified and the target is a symbolic link.

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

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

[EINTR] The **open**() operation was interrupted by a signal.

[EOPNOTSUPP] O_SHLOCK or O_EXLOCK is specified but the underlying file system does not support locking.

[EOPNOTSUPP] The named file is a special file mounted through a file system that does not support access to it (e.g. NFS).

[EWOULDBLOCK] O_NONBLOCK and one of O_SHLOCK or O_EXLOCK is specified and the file is locked.

[ENOSPC] O_CREAT is specified, the file does not exist, and the directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.

[ENOSPC] O_CREAT is specified, the file does not exist, and there are no free inodes on the file system on which the file is being created.

[EDQUOT] O_CREAT is specified, the file does not exist, and the directory in which the entry for the new file is being placed cannot be extended because the user’s quota of disk blocks on the file system containing the directory has been exhausted.

[EDQUOT] O_CREAT is specified, the file does not exist, and the user’s quota of inodes on the file system on which the file is being created has been exhausted.

[EIO] An I/O error occurred while making the directory entry or allocating the inode for O_CREAT.

[EINTEGRITY] Corrupted data was detected while reading from the file system.

[ETXTBSY] The file is a pure procedure (shared text) file that is being executed and the **open**() system call requests write access.

[EFAULT] The *path* argument points outside the process's allocated address space.

[EEXIST] O_CREAT and O_EXCL were specified and the file exists.

[EOPNOTSUPP] An attempt was made to open a socket (not currently implemented).

[EINVAL] An attempt was made to open a descriptor with an illegal combination of O_RDONLY, O_WRONLY, or O_RDWR, and O_EXEC or O_SEARCH.

[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 searching.

[ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a file descriptor associated with a directory.

[ENOTDIR] O_DIRECTORY is specified and the file is not a directory.

[ECAPMODE] AT_FDCWD is specified and the process is in capability mode.

[ECAPMODE] **open**() was called and the process is in capability mode.

[ENOTCAPABLE] *path* is an absolute path and the process is in capability mode.

[ENOTCAPABLE] *path* is an absolute path and O_RESOLVE_BENEATH is specified.

[ENOTCAPABLE] *path* contains a ".." component leading to a directory outside of the directory hierarchy specified by *fd* and the process is in capability mode.

[ENOTCAPABLE] *path* contains a ".." component leading to a directory outside of the directory hierarchy specified by *fd* and O_RESOLVE_BENEATH is specified.

[ENOTCAPABLE] *path* contains a ".." component, the vfs.lookup_cap_dotdot sysctl(3) is set, and the process is in capability mode.

## SEE ALSO

chmod(2), close(2), dup(2), fexecve(2), fhopen(2), getdtablesize(2), getfh(2), lgetfh(2), lseek(2), read(2), umask(2), write(2), fopen(3), capsicum(4)

## STANDARDS

These functions are specified by IEEE Std 1003.1-2008 ("POSIX.1"). FreeBSD sets *errno* to EMLINK instead of ELOOP as specified by POSIX when O_NOFOLLOW is set in flags and the final component of pathname is a symbolic link to distinguish it from the case of too many symbolic link traversals in one of its non-final components.

## HISTORY

The **open**() function appeared in Version 1 AT&T UNIX. The **openat**() function was introduced in FreeBSD 8.0. O_DSYNC appeared in 13.0.

## BUGS

The Open Group Extended API Set 2 specification requires that the test for whether *fd* is searchable is based on whether *fd* is open for searching, not whether the underlying directory currently permits searches. The present implementation of the *openat* checks the current permissions of directory instead.

The *mode* argument is variadic and may result in different calling conventions than might otherwise be expected.