open() - Unix, Linux System Call

Previous Page | Next Page

NAME

open, creat - open and possibly create a file or device

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>


int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
```

DESCRIPTION

Given a *pathname* for a file, **open**() returns a file descriptor, a small, non-negative integer for use in subsequent system calls (**read**(2), **write**(2), **lseek**(2), **fcntl**(2), etc.). The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

The new file descriptor is set to remain open across an **execve**(2) (i.e., the **FD_CLOEXEC** file descriptor flag described in **fcntl**(2) is initially disabled). The file offset is set to the beginning of the file (see **lseek**(2)).

A call to **open**() creates a new *open file description*, an entry in the system-wide table of open files. This entry records the file offset and the file status flags (modifiable via the **fcntl**() **F_SETFL** operation). A file descriptor is a reference to one of these entries; this reference is unaffected if *pathname* is subsequently removed or modified to refer to a different file. The new open file description is initially not shared with any other process, but sharing may arise via **fork**(2).

The parameter *flags* must include one of the following *access modes*: **O_RDONLY**, **O_WRONLY**, or **O_RDWR.** These request opening the file read-only, write-only, or read/write, respectively.

In addition, zero or more file creation flags and file status flags can be bitwise-*or*'d in *flags*. The *file creation flags* are **O_CREAT**, **O_EXCL**, **O_NOCTTY**, and **O_TRUNC**. The *file status flags* are all of the remaining flags listed below. The distinction between these two groups of flags is that the file status flags can be retrieved and (in some cases) modified using **fcntl**(2). The full list of file creation flags and file status flags is as follows:

Tag	Description
O_APPEND	
	The file is opened in append mode. Before each **write**(), the file offset is positioned at the end of the file, as if with **lseek**(). **O_APPEND** may lead to corrupted files on NFS file systems if more than one process appends data to a file at once. This is because NFS does not support appending to a file, so the client kernel has to simulate it, which can't be done without a race condition.
O_ASYNC	
	Enable signal-driven I/O: generate a signal (SIGIO by default, but this can be changed via **fcntl**(2)) when input or output becomes possible on this file descriptor. This feature is only available for terminals, pseudo-terminals, sockets, and (since Linux 2.6) pipes and FIFOs. See **fcntl**(2) for further details.
O_CREAT	
	If the file does not exist it will be created. The owner (user ID) of the file is set to the effective user ID of the process. The group ownership (group ID) is set either to the effective group ID of the process or to the group ID of the parent directory (depending on filesystem type and mount options, and the mode of the parent directory, see, e.g., the mount options *bsdgroups* and *sysvgroups* of the ext2 filesystem, as described in **mount**(8)).
O_DIRECT	
	Try to minimize cache effects of the I/O to and from this file. In general this will degrade performance, but it is useful in special situations, such as when applications do their own caching. File I/O is done directly to/from user space buffers. The I/O is synchronous, i.e., at the completion of a **read**(2) or **write**(2), data is guaranteed to have been transferred. Under Linux 2.4 transfer sizes, and the alignment of user buffer and file offset must all be multiples of the logical block size of the file system. Under Linux 2.6 alignment must fit the block size of the device. A semantically similar (but deprecated) interface for block devices is described in **raw**(8).
O_DIRECTORY	
	If *pathname* is not a directory, cause the open to fail. This flag is Linux-specific, and was added in kernel version 2.1.126, to avoid denial-of-service problems if **opendir**(3) is called on a FIFO or tape device, but should not be used outside of the implementation of **opendir**.

O_EXCL	When used with **O_CREAT**, if the file already exists it is an error and the **open**() will fail. In this context, a symbolic link exists, regardless of where it points to. **O_EXCL** is broken on NFS file systems; programs which rely on it for performing locking tasks will contain a race condition. The solution for performing atomic file locking using a lockfile is to create a unique file on the same file system (e.g., incorporating hostname and pid), use **link**(2) to make a link to the lockfile. If **link**() returns 0, the lock is successful. Otherwise, use **stat**(2) on the unique file to check if its link count has increased to 2, in which case the lock is also successful.
O_LARGEFILE	
	(LFS) Allow files whose sizes cannot be represented in an *off_t* (but can be represented in an *off64_t*) to be opened.
O_NOATIME	
	(Since Linux 2.6.8) Do not update the file last access time (st_atime in the inode) when the file is **read**(2). This flag is intended for use by indexing or backup programs, where its use can significantly reduce the amount of disk activity. This flag may not be effective on all filesystems. One example is NFS, where the server maintains the access time.
O_NOCTTY	
	If *pathname* refers to a terminal device — see **tty**(4) — it will not become the process's controlling terminal even if the process does not have one.
O_NOFOLLOW	
	If *pathname* is a symbolic link, then the open fails. This is a FreeBSD extension, which was added to Linux in version 2.1.126. Symbolic links in earlier components of the pathname will still be followed.
O_NONBLOCK or **O_NDELAY**	
	When possible, the file is opened in non-blocking mode. Neither the **open**() nor any subsequent operations on the file descriptor which is returned will cause the calling process to wait. For the handling of FIFOs (named pipes), see also **fifo**(7). For a discussion of the effect of **O_NONBLOCK** in conjunction with mandatory file locks and with file leases, see **fcntl**(2).
O_SYNC	The file is opened for synchronous I/O. Any **write**()s on the resulting file descriptor will block the calling process until the data has been physically written to the underlying hardware. *But see RESTRICTIONS below.*
O_TRUNC	
	If the file already exists and is a regular file and the open mode allows writing (i.e., is O_RDWR or O_WRONLY) it will be truncated to length 0. If the file is a FIFO or terminal device file, the O_TRUNC flag is ignored. Otherwise the effect of O_TRUNC is unspecified.
Some of these optional flags can be altered using **fcntl**() after the file has been opened. The argument *mode* specifies the permissions to use in case a new file is created. It is modified by the process's **umask** in the usual way: the permissions of the created file are **(mode & ~umask)**. Note that this mode only applies to future accesses of the newly created file; the **open**() call that creates a read-only file may well return a read/write file descriptor.	
The following symbolic constants are provided for *mode*:	
S_IRWXU	
	00700 user (file owner) has read, write and execute permission
S_IRUSR	
	00400 user has read permission
S_IWUSR	
	00200 user has write permission
S_IXUSR	
	00100 user has execute permission
S_IRWXG	
	00070 group has read, write and execute permission
S_IRGRP	
	00040 group has read permission
S_IWGRP	
	00020 group has write permission
S_IXGRP	
	00010 group has execute permission
S_IRWXO	
	00007 others have read, write and execute permission
S_IROTH	
	00004 others have read permission
S_IWOTH	
	00002 others have write permission
S_IXOTH	
	00001 others have execute permission

mode must be specified when **O_CREAT** is in the *flags*, and is ignored otherwise.

creat() is equivalent to **open**() with *flags* equal to **O_CREAT|O_WRONLY|O_TRUNC**.

RETURN VALUE

open() and **creat**() return the new file descriptor, or -1 if an error occurred (in which case, *errno* is set appropriately).

NOTES

Note that **open**() can open device special files, but **creat**() cannot create them; use **mknod**(2) instead.

On NFS file systems with UID mapping enabled, **open**() may return a file descriptor but e.g. **read**(2) requests are denied with **EACCES**. This is because the client performs **open**() by checking the permissions, but UID mapping is performed by the server upon read and write requests.

If the file is newly created, its st_atime, st_ctime, st_mtime fields (respectively, time of last access, time of last status change, and time of last modification; see **stat**(2)) are set to the current time, and so are the st_ctime and st_mtime fields of the parent directory. Otherwise, if the file is modified because of the O_TRUNC flag, its st_ctime and st_mtime fields are set to the current time.

ERRORS

Tag	Description
EACCES	The requested access to the file is not allowed, or search permission is denied for one of the directories in the path prefix of *pathname*, or the file did not exist yet and write access to the parent directory is not allowed. (See also **path_resolution**(2).)
EEXIST	*pathname* already exists and **O_CREAT** and **O_EXCL** were used.
EFAULT	*pathname* points outside your accessible address space.
EISDIR	*pathname* refers to a directory and the access requested involved writing (that is, **O_WRONLY** or **O_RDWR** is set).
ELOOP	Too many symbolic links were encountered in resolving *pathname*, or **O_NOFOLLOW** was specified but *pathname* was a symbolic link.
EMFILE	The process already has the maximum number of files open.
ENAMETOOLONG	
	pathname was too long.
ENFILE	The system limit on the total number of open files has been reached.
ENODEV	*pathname* refers to a device special file and no corresponding device exists. (This is a Linux kernel bug; in this situation ENXIO must be returned.)
ENOENT	O_CREAT is not set and the named file does not exist. Or, a directory component in *pathname* does not exist or is a dangling symbolic link.
ENOMEM	Insufficient kernel memory was available.
ENOSPC	*pathname* was to be created but the device containing *pathname* has no room for the new file.
ENOTDIR	
	A component used as a directory in *pathname* is not, in fact, a directory, or **O_DIRECTORY** was specified and *pathname* was not a directory.
ENXIO	O_NONBLOCK | O_WRONLY is set, the named file is a FIFO and no process has the file open for reading. Or, the file is a device special file and no corresponding device exists.
EOVERFLOW	
	pathname refers to a regular file, too large to be opened; see O_LARGEFILE above.
EPERM	The **O_NOATIME** flag was specified, but the effective user ID of the caller did not match the owner of the file and the caller was not privileged (**CAP_FOWNER**).
EROFS	*pathname* refers to a file on a read-only filesystem and write access was requested.
ETXTBSY	
	pathname refers to an executable image which is currently being executed and write access was requested.
EWOULDBLOCK	
	The **O_NONBLOCK** flag was specified, and an incompatible lease was held on the file (see **fcntl**(2)).

NOTE

Under Linux, the O_NONBLOCK flag indicates that one wants to open but does not necessarily have the intention to read or write. This is typically used to open devices in order to get a file descriptor for use with **ioctl**(2).

CONFORMING TO

SVr4, 4.3BSD, POSIX.1-2001. The **O_NOATIME**, **O_NOFOLLOW**, and **O_DIRECTORY** flags are Linux-specific. One may have to define the **_GNU_SOURCE** macro to get their definitions.

The (undefined) effect of **O_RDONLY | O_TRUNC** varies among implementations. On many systems the file is actually truncated.

The **O_DIRECT** flag was introduced in SGI IRIX, where it has alignment restrictions similar to those of Linux 2.4. IRIX has also a fcntl(2) call to query appropriate alignments, and sizes. FreeBSD 4.x introduced a flag of same name, but without alignment restrictions. Support was added under Linux in kernel version 2.4.10. Older Linux kernels simply ignore this flag. One may have to define the **_GNU_SOURCE** macro to get its definition.

BUGS

"The thing that has always disturbed me about O_DIRECT is that the whole interface is just stupid, and was probably designed by a deranged monkey on some serious mind-controlling substances." — Linus

Currently, it is not possible to enable signal-driven I/O by specifying **O_ASYNC** when calling **open**(); use **fcntl**(2) to enable this flag.

RESTRICTIONS

There are many infelicities in the protocol underlying NFS, affecting amongst others **O_SYNC** and **O_NDELAY**.

POSIX provides for three different variants of synchronised I/O, corresponding to the flags **O_SYNC**, **O_DSYNC** and **O_RSYNC**. Currently (2.1.130) these are all synonymous under Linux.

SEE ALSO

- close (2)
- dup (2)
- fcntl (2)
- link (2)
- lseek (2)
- mknod (2)
- mount (2)
- mmap (2)
- openat (2)
- path_resolution (2)
- read (2)
- socket (2)
- stat (2)
- umask (2)
- unlink (2)
- write (2)

Previous Page | Next Page Printer friendly page