NAME

read − read from a file descriptor

SYNOPSIS

#include <unistd.h>

ssize_t read(int *fd***, void ****buf***, size_t** *count***);**

DESCRIPTION

read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

If *count* is zero, **read**() returns zero and has no other results. If *count* is greater than **SSIZE_MAX**, the result is unspecified.

RETURN VALUE

On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. It is not an error if this number is smaller than the number of bytes requested; this may happen for example because fewer bytes are actually available right now (maybe because we were close to end-of-file, or because we are reading from a pipe, or from a terminal), or because **read**() was interrupted by a signal. On error, −1 is returned, and *errno* is set appropriately. In this case it is left unspecified whether the file position (if any) changes.

ERRORS

EAGAIN

The file descriptor *fd* refers to a file other than a socket and has been marked non-blocking (**O_NONBLOCK**), and the read would block.

EAGAIN or **EWOULDBLOCK**

The file descriptor *fd* refers to a socket and has been marked non-blocking (**O_NONBLOCK**), and the read would block. POSIX.1-2001 allows either error to be returned for this case, and does not require these constants to have the same value, so a portable application should check for both possibilities.

EBADF

fd is not a valid file descriptor or is not open for reading.

EFAULT

buf is outside your accessible address space.

EINTR

The call was interrupted by a signal before any data was read; see **signal**(7).

EINVAL

fd is attached to an object which is unsuitable for reading; or the file was opened with the **O_DIRECT** flag, and either the address specified in *buf*, the value specified in *count*, or the current file offset is not suitably aligned.

EINVAL

fd was created via a call to **timerfd_create**(2) and the wrong size buffer was given to **read**(); see **timerfd_create**(2) for further information.

EIO I/O error. This will happen for example when the process is in a background process group, tries to read from its controlling tty, and either it is ignoring or blocking **SIGTTIN** or its process group is orphaned. It may also occur when there is a low-level I/O error while reading from a disk or tape.

EISDIR

fd refers to a directory.

Other errors may occur, depending on the object connected to *fd*. POSIX allows a **read**() that is interrupted after reading some data to return −1 (with *errno* set to **EINTR**) or to return the number of bytes already read.

CONFORMING TO

SVr4, 4.3BSD, POSIX.1-2001.

NOTES

On NFS file systems, reading small amounts of data will only update the timestamp the first time, subsequent calls may not do so. This is caused by client side attribute caching, because most if not all NFS clients leave st_atime (last file access time) updates to the server and client side reads satisfied from the client's cache will not cause st_atime updates on the server as there are no server side reads. Unix semantics can be obtained by disabling client side attribute caching, but in most situations this will substantially increase server load and decrease performance.

Many file systems and disks were considered to be fast enough that the implementation of **O_NONBLOCK** was deemed unnecessary. So, **O_NONBLOCK** may not be available on files and/or disks.

SEE ALSO

close(2), **fcntl**(2), **ioctl**(2), **lseek**(2), **open**(2), **pread**(2), **readdir**(2), **readlink**(2), **readv**(2), **select**(2), **write**(2), **fread**(3)

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.