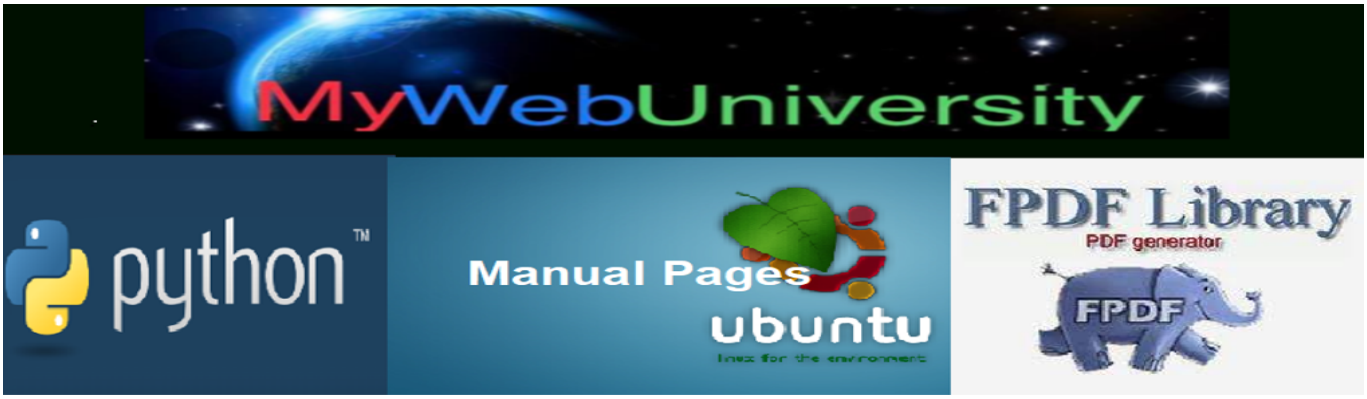

Rocky Enterprise Linux 9.2 Manual Pages on command 'read.2'

$ man read.2

READ(2) Linux Programmer's Manual READ(2)

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.

On files that support seeking, the read operation commences at the file offset, and the file offset is incremented by the number of bytes read. If the file offset is at or past the end of file, no bytes are read, and read() returns zero.

If count is zero, read() may detect the errors described below. In the absence of any er? rors, or if read() does not check for errors, a read() with a count of 0 returns zero and has no other effects.

According to POSIX.1, if count is greater than SSIZE_MAX, the result is implementation-de? fined; see NOTES for the upper limit on Linux.

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 sig?

nal. See also NOTES.

On error, -1 is returned, and errno is set appropriately. In this case, it is left un?
specified 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 nonblocking (O_NONBLOCK), and the read would block. See open(2) for further de? tails on the O_NONBLOCK flag.

EAGAIN or EWOULDBLOCK

The file descriptor fd refers to a socket and has been marked nonblocking (O_NON? BLOCK), 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 speci? fied in count, or the 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 terminal, and either it is ignor? ing 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. A further possi? ble cause of EIO on networked filesystems is when an advisory lock had been taken out on the file descriptor and this lock has been lost. See the Lost locks section of fcntl(2) for further details.

EISDIR fd refers to a directory.

Other errors may occur, depending on the object connected to fd.

CONFORMING TO

SVr4, 4.3BSD, POSIX.1-2001.

NOTES

The types size_t and ssize_t are, respectively, unsigned and signed integer data types

specified by POSIX.1.

On Linux, read() (and similar system calls) will transfer at most 0x7ffff000 (2,147,479,552) bytes, returning the number of bytes actually transferred. (This is true on both 32-bit and 64-bit systems.)

On NFS filesystems, reading small amounts of data will update the timestamp only 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.

BUGS

According to POSIX.1-2008/SUSv4 Section XSI 2.9.7 ("Thread Interactions with Regular File Operations"):

> All of the following functions shall be atomic with respect to each other in the ef?
> fects specified in POSIX.1-2008 when they operate on regular files or symbolic links:
> ...

Among the APIs subsequently listed are read() and readv(2). And among the effects that should be atomic across threads (and processes) are updates of the file offset. However, on Linux before version 3.14, this was not the case: if two processes that share an open file description (see open(2)) perform a read() (or readv(2)) at the same time, then the I/O operations were not atomic with respect updating the file offset, with the result that the reads in the two processes might (incorrectly) overlap in the blocks of data that they obtained. This problem was fixed in Linux 3.14.

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 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.