System Calls read(2)

NAME

read, readv, pread - read from file

SYNOPSIS

```
#include <unistd.h>

ssize_t read(int fildes, void *buf, size_t nbyte);

ssize_t pread(int fildes, void *buf, size_t nbyte, off_t
offset);

#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov, int
iovcnt);
```

DESCRIPTION

The read() function attempts to read nbyte bytes from the file associated with the open file descriptor, fildes, into the buffer pointed to by buf.

If nbyte is 0, read() returns 0 and has no other results.

On files that support seeking (for example, a regular file), the read() starts at a position in the file given by the file offset associated with fildes. The file offset is incremented by the number of bytes actually read.

Files that do not support seeking (for example, terminals) always read from the current position. The value of a file offset associated with such a file is undefined.

If fildes refers to a socket, read() is equivalent to recv(3SOCKET) with no flags set.

No data transfer will occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent read() requests is implementation-dependent.

When attempting to read from a regular file with mandatory file/record locking set (see chmod(2)), and there is a write lock owned by another process on the segment of the file to be read:

- If O_NDELAY or O_NONBLOCK is set, read() returns -1 and sets errno to EAGAIN.
- If O_NDELAY and O_NONBLOCK are clear, read() sleeps until the blocking record lock is removed.

When attempting to read from an empty pipe (or FIFO):

- If no process has the pipe open for writing, read() returns 0 to indicate end-of-file.
- If some process has the pipe open for writing and O_NDELAY is set, read() returns 0.
- If some process has the pipe open for writing and O_NONBLOCK is set, read() returns -1 and sets errno to EAGAIN.
- If O_NDELAY and O_NONBLOCK are clear, read() blocks until data is written to the pipe or the pipe is closed by all processes that had opened the pipe for writing.

When attempting to read a file associated with a terminal that has no data currently available:

- If O_NDELAY is set, read() returns 0.
- If O_NONBLOCK is set, read() returns -1 and sets errno to EAGAIN.

- If O_NDELAY and O_NONBLOCK are clear, read() blocks until data become available.

When attempting to read a file associated with a socket or a stream that is not a pipe, a FIFO, or a terminal, and the file has no data currently available:

- If O_NDELAY or O_NONBLOCK is set, read() returns -1 and sets errno to EAGAIN.

- If O_NDELAY and O_NONBLOCK are clear, read() blocks until data becomes available.

The read() function reads data previously written to a file. If any portion of a regular file prior to the end-of-file has not been written, read() returns bytes with value 0. For example, lseek(2) allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads in the gap between the previous end of data and the newly written data will return bytes with value 0 until data is written into the gap.

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with fildes.

Upon successful completion, where nbyte is greater than 0, read() will mark for update the st_atime field of the file, and return the number of bytes read. This number will never be greater than nbyte. The value returned may be less than nbyte if the number of bytes left in the file is less than nbyte, if the read() request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than nbyte bytes immediately available for reading. For example, a read() from a file associated with a terminal may return one typed line of data.

If a read() is interrupted by a signal before it reads any data, it will return -1 with errno set to EINTR.

If a read() is interrupted by a signal after it has successfully read some data, it will return the number of bytes read.

A read() from a STREAMS file can read data in three different modes: byte-stream mode, message-nondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the I_SRDOPT ioctl(2) request, and can be tested with the I_GRDOPT ioctl(). In byte-stream mode, read() retrieves data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, read() retrieves data until as many bytes as were requested are transferred, or until a message boundary is reached. If read() does not retrieve all the data in a message, the remaining data is left on the STREAM, and can be retrieved by the next read() call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the read() returns is discarded, and is not available for a subsequent read(), readv() or getmsg(2) call.

How read() handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, read() accepts data until it has read nbyte bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The read() function then returns the number of bytes read, and places the zero-byte message back on the STREAM to be retrieved by the next read(), readv() or getmsg(2). In message-nondiscard mode or message-discard mode, a zero-byte message returns 0 and the message is removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message is removed from the STREAM and 0 is returned, regardless of the read mode.

A read() from a STREAMS file returns the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMs are in control-normal mode, in which a read() from a STREAMS file can only process messages that contain a data part but do not contain a control part. The read() fails if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the I_SRDOPT ioctl() command. In control-data mode, read() converts any control part to data and passes it to the application before passing any data part originally present in the same message. In control-discard mode, read() discards message control parts but returns to the process any data part in the message.

In addition, read() and readv() will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of errno does not reflect the result of read() or readv() but reflects the prior error. If a hangup occurs on the STREAM being read, read() continues to operate normally until the STREAM head read queue is empty. Thereafter, it returns 0.

readv()

The readv() function is equivalent to read(), but places the input data into the iovcnt buffers specified by the members of the iov array: iov0, iov1, ..., iov[iovcnt-1]. The iovcnt argument is valid if greater than 0 and less than or equal to IOV_MAX.

The iovec structure contains the following members:

```
caddr_t   iov_base;
int       iov_len;
```

Each iovec entry specifies the base address and length of an area in memory where data should be placed. The readv() function always fills an area completely before proceeding to the next.

Upon successful completion, readv() marks for update the st_atime field of the file.

pread()

The pread() function performs the same action as read(), except that it reads from a given position in the file without changing the file pointer. The first three arguments to pread() are the same as read() with the addition of a fourth argument offset for the desired position inside the file. pread() will read up to the maximum offset value that can be represented in an off_t for regular files. An attempt to perform a pread() on a file that is incapable of seeking results in an error.

RETURN VALUES

Upon successful completion, read() and readv() return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return -1 and set errno to indicate the error.

ERRORS

The read(), readv(), and pread() functions will fail if:

EAGAIN
: Mandatory file/record locking was set, O_NDELAY or O_NONBLOCK was set, and there was a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; no data is waiting to be read on a file associated with a tty device and O_NONBLOCK was set; or no message is waiting to be read on a stream and O_NDELAY or O_NONBLOCK was set.

EBADF
: The fildes argument is not a valid file descriptor open for reading.

EBADMSG
Message waiting to be read on a stream is not a data message.

EDEADLK
The read was going to go to sleep and cause a deadlock to occur.

EINTR A signal was caught during the read operation and no data was transferred.

EINVAL
An attempt was made to read from a stream linked to a multiplexor.

EIO A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group of the process is orphaned.

EISDIR
The fildes argument refers to a directory on a file system type that does not support read operations on directories.

ENOLCK
The system record lock table was full, so the read() or readv() could not go to sleep until the blocking record lock was removed.

ENOLINK
The fildes argument is on a remote machine and the link to that machine is no longer active.

ENXIO The device associated with fildes is a block special or character special file and the value of the file pointer is out of range.

The read() and pread() functions will fail if:

EFAULT
The buf argument points to an illegal address.

EINVAL
The nbyte argument overflowed an ssize_t.

The read() and readv() functions will fail if:

EOVERFLOW
The file is a regular file, nbyte is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with fildes.

The readv() function may fail if:

EFAULT
The iov argument points outside the allocated address space.

EINVAL
The iovcnt argument was less than or equal to 0 or greater than {IOV_MAX}. (See intro(2) for a definition of {IOV_MAX}).

EINVAL
One of the iov_len values in the iov array was negative, or the the sum of the iov_len values in the iov array overflowed an ssize_t.

The pread() function will fail and the file pointer remain unchanged if:

ESPIPE
The fildes argument is associated with a pipe or FIFO.

USAGE

The pread() function has a transitional interface for 64-bit file offsets. See lf64(5).

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	read() is Async-Signal-Safe

SEE ALSO

intro(2), chmod(2), creat(2), dup(2), fcntl(2), getmsg(2), ioctl(2), lseek(2), open(2), pipe(2), recv(3SOCKET), attributes(5), lf64(5), streamio(7I), termio(7I)

SunOS 5.9 Last change: 7 May 2001