NAME

read, pread, readv – read from file

SYNOPSIS

#include <sys/types.h>
#include <sys/uio.h>
#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***);**

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

MT-LEVEL

read( ) is Async-Signal-Safe

DESCRIPTION

read( ) attempts to read *nbyte* bytes from the file associated with *fildes* into the buffer pointed to by *buf*. If *nbyte* is zero, **read( )** returns zero and has no other results. *fildes* is an open file descriptor.

On devices capable of seeking, the **read( )** starts at a position in the file given by the file pointer associated with *fildes*. On return from **read( )**, the file pointer is incremented by the number of bytes actually read.

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

pread( ) 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. An attempt to perform a **pread( )** on a file that is incapable of seeking results in an error.

readv( ) performs the same action as **read( )**, but places the input data into the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], . . ., *iov[iovcnt*– 1].

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. **readv( )** always fills one buffer completely before proceeding to the next.

On success, **read( )** and **readv( )** return the number of bytes actually read and placed in the buffer; this number may be less than *nbyte* if the file is associated with a communication line (see **ioctl**(2) and **termio**(7I)), or if the number of bytes left in the file is less than *nbyte* , or if the file is a pipe or a special file. A value of **0** is returned when an end-of-file has been reached.

read( ) reads data previously written to a file. If any portion of an ordinary file prior to the end of file has not been written, **read( )** returns the number of bytes read as **0**. For example, the **lseek** routine allows the file pointer to be set beyond the end of existing data in the file. If additional data is written at this point, subsequent reads in the gap between the previous end of data and newly written data return bytes with a value of **0** until data is written into the gap.

A **read( )** or **readv( )** from a STREAMS (see **intro**(2)) file can operate 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 (see **streamio**(7I)), and can be tested with the **I_GRDOPT ioctl**(2) request.

In byte-stream mode, **read( )** and **readv( )** retrieve data from the stream until they have retrieved *nbyte* bytes, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, **read( )** and **readv( )** retrieve data until they have read *nbyte* bytes, or until they reach a message boundary. If **read( )** or **readv( )** does not retrieve all the data in a message, the remaining data is replaced on the stream and can be retrieved by the next **read( )** or **readv( )** call. Message-discard mode also retrieves data until it has retrieved *nbyte* bytes, or it reaches a message boundary. However, unread data remaining in a message after the **read** or **readv** returns is discarded, and is not available for a subsequent **read( )**, **readv( )**, or **getmsg( )** (see **getmsg**(2)).

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 stream that is not a pipe or FIFO, or terminal, and that 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.

When reading from a STREAMS file, handling of zero-byte 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. **read( )** then returns the number of bytes read, and places the zero-byte message back on the stream to be retrieved by the next **read( )** or **getmsg( )** (see **getmsg**(2)). In the two other modes, a zero-byte message returns a value of **0** and the message is removed from the stream. When a zero-byte message is read as the first message on a stream, a value of **0** is returned regardless of the **read( )** mode.

A **read( )** or **readv( )** 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.

Normally, a **read( )** from a STREAMS file can only process messages with data and without control information. The **read( )** fails if a message containing control information 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**(2). In control-data mode, control messages are converted to data messages by **read( )**. In control-discard mode, control messages are discarded by **read( )**, but any data associated with the control messages is returned to the user.

RETURN VALUES

On success a non-negative integer is returned indicating the number of bytes actually read. Otherwise, a −1 is returned and **errno** is set to indicate the error.

ERRORS

read( ), **pread( )**, and **readv( )** fail if one or more of the following are true:

EAGAIN Mandatory file/record locking was set, **O_NDELAY** or **O_NONBLOCK** was set, and there was a blocking record lock.

EAGAIN Total amount of system memory available when reading using raw I/O is temporarily insufficient.

EAGAIN No data is waiting to be read on a file associated with a tty device and **O_NONBLOCK** was set.

EAGAIN No message is waiting to be read on a stream and **O_NDELAY** or **O_NONBLOCK** was set.

EBADF *fildes* 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.

EFAULT *buf* points to an illegal address.

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

EINVAL Attempted 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 *fildes* 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 *fildes* 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.

In addition, **readv( )** may return one of the following errors:

EFAULT *iov* points outside the allocated address space.

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

EINVAL The sum of the **iov_len** values in the *iov* array overflowed an int.

In addition, **pread( )** fails and the file pointer remains unchanged if the following is true:

ESPIPE *fildes* is associated with a pipe or fifo.

A **read( )** from a STREAMS file also fails if an error message is received at the stream head. In this case, **errno** is set to the value returned in the error message. 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.**

SEE ALSO

intro(2), **chmod**(2), **creat**(2), **dup**(2), **fcntl**(2), **getmsg**(2), **ioctl**(2), **open**(2), **pipe**(2), **streamio**(7I), **termio**(7I)