NAME

write, pwrite, writev − write on a file

SYNOPSIS

#include <unistd.h>

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

#include <sys/types.h>
#include <unistd.h>

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

#include <sys/types.h>
#include <sys/uio.h>

int writev(int *fildes***, const struct iovec ****iov***, int** *iovcnt***);**

MT-LEVEL

write( ) is Async-Signal-Safe

DESCRIPTION

write( ) attempts to write *nbyte* bytes from the buffer pointed to by *buf* to the file descriptor specified by *fildes*. If *nbyte* is zero and the file is a regular file, **write( )** returns zero and has no other results.

pwrite( ) performs the same action as **write( )**, except that it writes into a given position without changing the file pointer. The first three arguments to **pwrite( )** are the same as **write( )** with the addition of a fourth argument *offset* for the desired position inside the file.

writev( ) performs the same action as **write( )**, but gathers the output data from the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt* − 1]. The *iovcnt* buffer is valid if greater than 0 and less than or equal to **{IOV_MAX}**. (See **intro**(2) for a definition of **{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 from which data should be written. **writev( )** always writes all data from an area before proceeding to the next.

On devices capable of seeking, the actual writing of data starts at the position in the file indicated by the file pointer. On return from **write( )**, the file pointer is incremented by the number of bytes actually written. On a regular file, if the incremented file pointer is greater than the length of the file, the length of the file is set to the new file pointer.

On devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

If the **O_APPEND** flag of the file status flags is set, the file pointer is set to the end of the file prior to each **write( )**. The system guarantees that no intervening file modification operation will occur between changing the file offset and the write operation.

For regular files, if the **O_SYNC** flag of the file status flags is set, **write( )** does not return until both the file data and file status have been physically updated. This function is for special applications that require extra reliability at the cost of performance. For block special files, if **O_SYNC** is set, **write( )** does not return until the data has been physically updated.

A **write( )** to a regular file is blocked if mandatory file/record locking is set (see **chmod**(2)), and there is a record lock owned by another process on the segment of the file to be written:

- If **O_NDELAY** or **O_NONBLOCK** is set, **write( )** returns **−1** and sets **errno** to **EAGAIN**.
- If **O_NDELAY** and **O_NONBLOCK** are clear, **write( )** sleeps until all blocking locks are removed or the **write( )** is terminated by a signal.

If a **write( )** requests that more bytes be written than there is room for—for example, if the write would exceed the process file size limit (see **getrlimit**(2) and **ulimit**(2)), the system file size limit, or the free space on the device—only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A **write( )** of 512-bytes returns 20. The next **write( )** of a non-zero number of bytes gives a failure return (except as noted for pipes and FIFO below).

Write requests to a pipe or FIFO are handled the same as a regular file with the following exceptions:

- There is no file offset associated with a pipe, hence each write request appends to the end of the pipe.
- Write requests of **{PIPE_BUF}** bytes or less are guaranteed not to be interleaved with data from other processes doing writes on the same pipe. Writes of greater than **{PIPE_BUF}** bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the **O_NONBLOCK** or **O_NDELAY** flags are set.
- If **O_NONBLOCK** and **O_NDELAY** are clear, a write request may cause the process to block, but on normal completion it returns *nbyte*.
- If **O_NONBLOCK** and **O_NDELAY** are set, **write( )** does not block the process. If a **write( )** request for **{PIPE_BUF}** or fewer bytes succeeds completely **write( )** returns *nbyte*. Otherwise, if **O_NONBLOCK** is set, it returns **−1** and sets **errno** to **EAGAIN** or if **O_NDELAY** is set, it returns **0**. A **write( )** request for greater than **{PIPE_BUF}** bytes transfers what it can and returns the number of bytes written or it transfers no data and, if **O_NONBLOCK** is set, returns **−1** with **errno** set to **EAGAIN** or if **O_NDELAY** is set, it returns **0**. Finally, if a request is greater than **{PIPE_BUF}** bytes and all data previously written to the pipe has been read, **write( )** transfers at least **{PIPE_BUF}** bytes.

When attempting to write to a file descriptor (other than a pipe, FIFO, or stream) that supports nonblocking writes and cannot accept the data immediately:

- If **O_NONBLOCK** and **O_NDELAY** are clear, **write( )** blocks until the data can be accepted.
- If **O_NONBLOCK** or **O_NDELAY** is set, **write( )** does not block the process. If some data can be written without blocking the process, **write( )** writes what it can and returns the number of bytes written. Otherwise, if **O_NONBLOCK** is set, it returns **−1** and sets **errno** to **EAGAIN** or if **O_NDELAY** is set, it returns **0**.

For STREAMS files (see **intro**(2) and **streamio**(7I)), the operation of **write( )** is determined by the values of the minimum and maximum *nbyte* range ("packet size") accepted by the stream. These values are contained in the topmost stream module, and can not be set or tested from user level. If *nbyte* falls within the packet size range, *nbyte* bytes are written. If *nbyte* does not fall within the range and the minimum packet size value is zero, **write( )** breaks the buffer into maximum packet size segments prior to sending the data downstream (the last segment may be smaller than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, **write( )** fails and sets **errno** to **ERANGE**. Writing a zero-length buffer (*nbyte* is zero) to a STREAMS device sends a zero length message with zero returned. However, writing a zero-length buffer to a pipe or FIFO sends no message and zero is returned. The user program may issue the **I_SWROPT ioctl**(2) to enable zero-length messages to be sent across the pipe or FIFO (see **streamio**(7I)).

When writing to a stream, data messages are created with a priority band of zero. When writing to a stream

that is not a pipe or FIFO:

- If **O_NDELAY** and **O_NONBLOCK** are not set, and the stream cannot accept data (the stream write queue is full due to internal flow control conditions), **write( )** blocks until data can be accepted.
- If **O_NDELAY** or **O_NONBLOCK** is set and the stream cannot accept data, **write( )** returns **-1** and sets **errno** to **EAGAIN**.
- If **O_NDELAY** or **O_NONBLOCK** is set and part of the buffer has already been written when a condition occurs in which the stream cannot accept additional data, **write( )** terminates and returns the number of bytes written.

RETURN VALUES

On success, **write( )** returns the number of bytes actually written. Otherwise, it returns **-1** and sets **errno** to indicate the error.

ERRORS

write( ), **pwrite( )**, and **writev( )** fail and the file pointer remains unchanged if one or more of the following are true:

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

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

An attempt is made to write to a stream that can not accept data with the **O_NDELAY** or **O_NONBLOCK** flag set.

If a write to a pipe or FIFO of **{PIPE_BUF}** bytes or less is requested and less than *nbytes* of free space is available.

EBADF *fildes* is not a valid file descriptor open for writing.

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

EDQUOT The user's quota of disk blocks on the file system containing the file has been exhausted.

EFAULT *buf* points to an illegal address.

EFBIG An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see **getrlimit**(2) and **ulimit**(2)).

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

EINVAL An attempt is made to write to a stream linked below a multiplexor.

EIO The process is in the background and is attempting to write to its controlling terminal whose **TOSTOP** flag is set; the process is neither ignoring nor blocking **SIGTTOU** signals, and the process group of the process is orphaned.

ENOLCK Enforced record locking was enabled and **{LOCK_MAX}** regions are already locked in the system.

The system record lock table was full, so the write 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.

ENOSPC During a write to an ordinary file, there is no free space left on the device.

ENOSR An attempt is made to write to a stream with insufficient STREAMS memory resources available in the system.

ENXIO A hangup occurred on the stream being written to.

EPIPE and **SIGPIPE** signal

An attempt is made to write to a pipe that is not open for reading by any process (or to a file descriptor created by **socket**(3N), using type **SOCK_STREAM** that is no longer connected to a peer endpoint). Note: an attempted write of this kind also causes you to receive a **SIGPIPE** signal from the kernel. If you've not made a special provision to catch or ignore this signal, then your process dies.

EPIPE An attempt is made to write to a FIFO that is not open for reading by any process.

An attempt is made to write to a pipe that has only one end open.

ERANGE An attempt is made to write to a stream with *nbyte* outside specified minimum and maximum write range, and the minimum value is non-zero.

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

EINVAL *iovcnt* was less than or equal to 0, or greater than {**IOV_MAX**}.

One of the **iov_len** values in the *iov* array was negative.

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

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

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

A **write( )** to a STREAMS file can fail if an error message has been received at the stream head. In this case, **errno** is set to the value included in the error message.

Upon successful completion **write( )** and **writev( )** mark for update the **st_ctime** and **st_mtime** fields of the file.

SEE ALSO

Intro(2), **chmod**(2), **creat**(2), **dup**(2), **fcntl**(2), **getrlimit**(2), **ioctl**(2), **lseek**(2), **open**(2), **pipe**(2), **ulimit**(2), **socket**(3N), **streamio**(7I)