Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'write.3p' command

$ man write.3p

WRITE(3P) POSIX Programmer's Manual WRITE(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pwrite, write ? write on a file

SYNOPSIS

```
#include <unistd.h>
ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
    off_t offset);
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

DESCRIPTION

The write() function shall attempt to write nbyte bytes from the buffer pointed to by buf to the file associated with the open file descriptor, fildes.

Before any action described below is taken, and if nbyte is zero and the file is a regular file, the write() function may detect and return errors as described below. In the absence of errors, or if error detec? tion is not performed, the write() function shall return zero and have no other results. If nbyte is zero and the file is not a regular file, the results are unspecified.

On a regular file or other file capable of seeking, the actual writing of data shall proceed from the position in the file indicated by the file offset associated with fildes. Before successful return from write(), the file offset shall be incremented by the number of bytes actually written. On a regular file, if the position of the last byte written is greater than or equal to the length of the file, the length of the file shall be set to this position plus one.

On a file not capable of seeking, writing shall always take place starting at the current position. The value of a file offset associated with such a device is undefined.

If the O_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file prior to each write and no inter? vening file modification operation shall occur between changing the file offset and the write operation.

If a write() requests that more bytes be written than there is room for (for example, the file size limit of the process or the physical end of a medium), only as many bytes as there is room for shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would give a failure return (except as noted below).

If the request would cause the file size to exceed the soft file size limit for the process and there is no room for any bytes to be written, the request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

If write() is interrupted by a signal before it writes any data, it shall return -1 with errno set to [EINTR].

If write() is interrupted by a signal after it successfully writes some data, it shall return the number of bytes written.

If the value of nbyte is greater than {SSIZE_MAX}, the result is imple? mentation-defined.

After a write() to a regular file has successfully returned:

* Any successful read() from each byte position in the file that was

modified by that write shall return the data specified by the write() for that position until such byte positions are again modi? fied.

* Any subsequent successful write() to the same byte position in the file shall overwrite that file data.

Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the following exceptions:

* There is no file offset associated with a pipe, hence each write request shall append to the end of the pipe.
* Write requests of {PIPE_BUF} bytes or less shall not 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 flag of the file status flags is set.
* If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on normal completion it shall return nbyte.
* If the O_NONBLOCK flag is set, write() requests shall be handled differently, in the following ways:
  -- The write() function shall not block the thread.
  -- A write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there is sufficient space available in the pipe, write() shall transfer all the data and return the number of bytes requested. Otherwise, write() shall transfer no data and return -1 with errno set to [EAGAIN].
  -- A write request for more than {PIPE_BUF} bytes shall cause one of the following:
    -- When at least one byte can be written, transfer what it can and return the number of bytes written. When all data pre? viously written to the pipe is read, it shall transfer at least {PIPE_BUF} bytes.
    -- When no data can be written, transfer no data, and return -1 with errno set to [EAGAIN].

When attempting to write to a file descriptor (other than a pipe or

FIFO) that supports non-blocking writes and cannot accept the data im?
mediately:

* If the O_NONBLOCK flag is clear, write() shall block the calling thread until the data can be accepted.
* If the O_NONBLOCK flag is set, write() shall not block the thread. If some data can be written without blocking the thread, write() shall write what it can and return the number of bytes written. Otherwise, it shall return -1 and set errno to [EAGAIN].

Upon successful completion, where nbyte is greater than 0, write() shall mark for update the last data modification and last file status change timestamps of the file, and if the file is a regular file, the S_ISUID and S_ISGID bits of the file mode may be cleared.

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

If fildes refers to a socket, write() shall be equivalent to send() with no flags set.

If the O_DSYNC bit has been set, write I/O operations on the file de? scriptor shall complete as defined by synchronized I/O data integrity completion.

If the O_SYNC bit has been set, write I/O operations on the file de? scriptor shall complete as defined by synchronized I/O file integrity completion.

If fildes refers to a shared memory object, the result of the write() function is unspecified.

If fildes refers to a typed memory object, the result of the write() function is unspecified.

If fildes refers to a STREAM, the operation of write() shall be deter? mined by the values of the minimum and maximum nbyte range (packet size) accepted by the STREAM. These values are determined by the top? most STREAM module. If nbyte falls within the packet size range, nbyte bytes shall be written. If nbyte does not fall within the range and the minimum packet size value is 0, write() shall break the buffer into maximum packet size segments prior to sending the data downstream (the

last segment may contain less than the maximum packet size). If nbyte does not fall within the range and the minimum value is non-zero, write() shall fail with errno set to [ERANGE]. Writing a zero-length buffer (nbyte is 0) to a STREAMS device sends 0 bytes with 0 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no message and 0 is returned. The process may issue I_SWROPT ioctl() to enable zero-length messages to be sent across the pipe or FIFO.

When writing to a STREAM, data messages are created with a priority band of 0. When writing to a STREAM that is not a pipe or FIFO:

* If O_NONBLOCK is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control condi? tions), write() shall block until data can be accepted.
* If O_NONBLOCK is set and the STREAM cannot accept data, write() shall return -1 and set errno to [EAGAIN].
* If O_NONBLOCK is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data oc? curs, write() shall terminate and return the number of bytes writ? ten.

In addition, write() shall fail if the STREAM head has processed an asynchronous error before the call. In this case, the value of errno does not reflect the result of write(), but reflects the prior error.

The pwrite() function shall be equivalent to write(), except that it writes into a given position and does not change the file offset (re? gardless of whether O_APPEND is set). 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. An attempt to perform a pwrite() on a file that is incapable of seeking shall result in an error.

RETURN VALUE

Upon successful completion, these functions shall return the number of bytes actually written to the file associated with fildes. This number shall never be greater than nbyte. Otherwise, -1 shall be returned and

errno set to indicate the error.

ERRORS

These functions shall fail if:

EAGAIN The file is neither a pipe, nor a FIFO, nor a socket, the O_NON? BLOCK flag is set for the file descriptor, and the thread would be delayed in the write() operation.

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

EFBIG An attempt was made to write a file that exceeds the implementa? tion-defined maximum file size or the file size limit of the process, and there was no room for any bytes to be written.

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

EINTR The write operation was terminated due to the receipt of a sig? nal, and no data was transferred.

EIO The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined condi? tions.

ENOSPC There was no free space remaining on the device containing the file.

ERANGE The transfer request size was outside the range supported by the STREAMS file associated with fildes.

The pwrite() function shall fail if:

EINVAL The file is a regular file or block special file, and the offset argument is negative. The file offset shall remain unchanged.

ESPIPE The file is incapable of seeking.

The write() function shall fail if:

EAGAIN The file is a pipe or FIFO, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the write

operation.

EAGAIN or EWOULDBLOCK

The file is a socket, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the write opera? tion.

ECONNRESET

A write was attempted on a socket that is not connected.

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal shall also be sent to the thread.

EPIPE A write was attempted on a socket that is shut down for writing, or is no longer connected. In the latter case, if the socket is of type SOCK_STREAM, a SIGPIPE signal shall also be sent to the thread.

These functions may fail if:

EINVAL The STREAM or multiplexer referenced by fildes is linked (di? rectly or indirectly) downstream from a multiplexer.

EIO A physical I/O error has occurred.

ENOBUFS

Insufficient resources were available in the system to perform the operation.

ENXIO A request was made of a nonexistent device, or the request was outside the capabilities of the device.

ENXIO A hangup occurred on the STREAM being written to.

A write to a STREAMS file may fail if an error message has been re? ceived at the STREAM head. In this case, errno is set to the value in? cluded in the error message.

The write() function may fail if:

EACCES A write was attempted on a socket and the calling process does not have appropriate privileges.

ENETDOWN

A write was attempted on a socket and the local network inter? face used to reach the destination is down.

ENETUNREACH

A write was attempted on a socket and no route to the network is present.

The following sections are informative.

EXAMPLES

Writing from a Buffer

The following example writes data from the buffer pointed to by buf to the file associated with the file descriptor fd.

```
#include <sys/types.h>
#include <string.h>
...
char buf[20];
size_t nbytes;
ssize_t bytes_written;
int fd;
...
strcpy(buf, "This is a test\n");
nbytes = strlen(buf);
bytes_written = write(fd, buf, nbytes);
...
```

APPLICATION USAGE

None.

RATIONALE

See also the RATIONALE section in read().

An attempt to write to a pipe or FIFO has several major characteris? tics:

* Atomic/non-atomic: A write is atomic if the whole amount written in one operation is not interleaved with data from any other process. This is useful when there are multiple writers sending data to a single reader. Applications need to know how large a write request can be expected to be performed atomically. This maximum is called {PIPE_BUF}. This volume of POSIX.1?2017 does not say whether write requests for more than {PIPE_BUF} bytes are atomic, but requires

that writes of {PIPE_BUF} or fewer bytes shall be atomic.

* Blocking/immediate: Blocking is only possible with O_NONBLOCK clear. If there is enough space for all the data requested to be written immediately, the implementation should do so. Otherwise, the calling thread may block; that is, pause until enough space is available for writing. The effective size of a pipe or FIFO (the maximum amount that can be written in one operation without block? ing) may vary dynamically, depending on the implementation, so it is not possible to specify a fixed value for it.

* Complete/partial/deferred: A write request:

```
int fildes;
size_t nbyte;
ssize_t ret;
char *buf;
ret = write(fildes, buf, nbyte);
```

may return:

Complete ret=nbyte

Partial ret<nbyte

This shall never happen if nbyte?{PIPE_BUF}. If it does happen (with nbyte>{PIPE_BUF}), this volume of POSIX.1?2017 does not guarantee atomicity, even if ret?{PIPE_BUF}, because atomicity is guaranteed according to the amount requested, not the amount written.

Deferred: ret=-1, errno=[EAGAIN]

This error indicates that a later request may succeed. It does not indicate that it shall succeed, even if nbyte?{PIPE_BUF}, because if no process reads from the pipe or FIFO, the write never succeeds. An application could usefully count the number of times [EAGAIN] is caused by a particular value of nbyte>{PIPE_BUF} and per? haps do later writes with a smaller value, on the assump? tion that the effective size of the pipe may have de? creased.

Partial and deferred writes are only possible with O_NONBLOCK set.

The relations of these properties are shown in the following tables:

??

? Write to a Pipe or FIFO with O_NONBLOCK clear ?

??

?Immediately Writable:? None Some nbyte ?

??

?nbyte?{PIPE_BUF} ?Atomic blocking Atomic blocking Atomic immediate ?

? ?nbyte nbyte nbyte ?

??

?nbyte>{PIPE_BUF} ?Blocking nbyte Blocking nbyte Blocking nbyte ?

??

If the O_NONBLOCK flag is clear, a write request shall block if the amount writable immediately is less than that requested. If the flag is set (by fcntl()), a write request shall never block.

??

? Write to a Pipe or FIFO with O_NONBLOCK set ?

??

?Immediately Writable:? None Some nbyte ?

??

?nbyte?{PIPE_BUF} ?-1, [EAGAIN] -1, [EAGAIN] Atomic nbyte ?

??

?nbyte>{PIPE_BUF} ?-1, [EAGAIN] <nbyte or -1, ?nbyte or -1, ?

? ? [EAGAIN] [EAGAIN] ?

??

There is no exception regarding partial writes when O_NONBLOCK is set.

With the exception of writing to an empty pipe, this volume of POSIX.1?2017 does not specify exactly when a partial write is performed since that would require specifying internal details of the implementa? tion. Every application should be prepared to handle partial writes when O_NONBLOCK is set and the requested amount is greater than {PIPE_BUF}, just as every application should be prepared to handle par? tial writes on other kinds of file descriptors.

The intent of forcing writing at least one byte if any can be written is to assure that each write makes progress if there is any room in the pipe. If the pipe is empty, {PIPE_BUF} bytes must be written; if not, at least some progress must have been made.

Where this volume of POSIX.1?2017 requires -1 to be returned and errno set to [EAGAIN], most historical implementations return zero (with the O_NDELAY flag set, which is the historical predecessor of O_NONBLOCK, but is not itself in this volume of POSIX.1?2017). The error indica? tions in this volume of POSIX.1?2017 were chosen so that an application can distinguish these cases from end-of-file. While write() cannot re? ceive an indication of end-of-file, read() can, and the two functions have similar return values. Also, some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that the reader should get an end-of-file indication; for those systems, a return value of zero from write() indicates a successful write of an end-of-file in? dication.

Implementations are allowed, but not required, to perform error check? ing for write() requests of zero bytes.

The concept of a {PIPE_MAX} limit (indicating the maximum number of bytes that can be written to a pipe in a single operation) was consid? ered, but rejected, because this concept would unnecessarily limit ap? plication writing.

See also the discussion of O_NONBLOCK in read().

Writes can be serialized with respect to other reads and writes. If a read() of file data can be proven (by any means) to occur after a write() of the data, it must reflect that write(), even if the calls are made by different processes. A similar requirement applies to mul? tiple write operations to the same file position. This is needed to guarantee the propagation of data from write() calls to subsequent read() calls. This requirement is particularly significant for net? worked file systems, where some caching schemes violate these seman? tics.

Note that this is specified in terms of read() and write(). The XSI

extensions readv() and writev() also obey these semantics. A new ``high-performance'' write analog that did not follow these serializa? tion requirements would also be permitted by this wording. This volume of POSIX.1?2017 is also silent about any effects of application-level caching (such as that done by stdio).

This volume of POSIX.1?2017 does not specify the value of the file off? set after an error is returned; there are too many cases. For program? ming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EA? GAIN], clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

This volume of POSIX.1?2017 does not specify the behavior of concurrent writes to a regular file from multiple threads, except that each write is atomic (see Section 2.9.7, Thread Interactions with Regular File Op? erations). Applications should use some form of concurrency control.

This volume of POSIX.1?2017 intentionally does not specify any pwrite() errors related to pipes, FIFOs, and sockets other than [ESPIPE].

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), creat(), dup(), fcntl(), getrlimit(), lseek(), open(), pipe(), read(), ulimit(), writev()

The Base Definitions volume of POSIX.1?2017, <limits.h>, <stropts.h>, <sys_uio.h>, <unistd.h>

COPYRIGHT

is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .