NAME

sendfile - send a file to a socket

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

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

int
sendfile(*int fd*, *int s*, *off_t offset*, *size_t nbytes*, *struct sf_hdtr *hdtr*, *off_t *sbytes*, *int flags*);

DESCRIPTION

The **sendfile**() system call sends a regular file or shared memory object specified by descriptor *fd* out a stream socket specified by descriptor *s*.

The *offset* argument specifies where to begin in the file. Should *offset* fall beyond the end of file, the system will return success and report 0 bytes sent as described below. The *nbytes* argument specifies how many bytes of the file should be sent, with 0 having the special meaning of send until the end of file has been reached.

An optional header and/or trailer can be sent before and after the file data by specifying a pointer to a *struct sf_hdtr*, which has the following structure:

```
struct sf_hdtr {
        struct iovec *headers;      /* pointer to header iovecs */
        int hdr_cnt;                /* number of header iovecs */
        struct iovec *trailers;     /* pointer to trailer iovecs */
        int trl_cnt;                /* number of trailer iovecs */
};
```

The *headers* and *trailers* pointers, if non-NULL, point to arrays of *struct iovec* structures. See the **writev**() system call for information on the iovec structure. The number of iovecs in these arrays is specified by *hdr_cnt* and *trl_cnt*.

If non-NULL, the system will write the total number of bytes sent on the socket to the variable pointed to by *sbytes*.

The least significant 16 bits of *flags* argument is a bitmap of these values:

SF_NODISKIO — This flag causes **sendfile** to return EBUSY instead of blocking when a busy page is encountered. This rare situation can happen if some other process is now working with the same region of the file. It is advised to retry the operation after a short period.

Note that in older FreeBSD versions the SF_NODISKIO had slightly different notion. The flag prevented **sendfile** to run I/O operations in case if an invalid (not cached) page is encountered, thus avoiding blocking on I/O. Starting with FreeBSD 11 **sendfile** sending files off the ffs(7) filesystem does not block on I/O (see *IMPLEMENTATION NOTES*), so the condition no longer applies. However, it is safe if an application utilizes SF_NODISKIO and on EBUSY performs the same action as it did in older FreeBSD versions, e.g., aio_read(2), read(2) or **sendfile** in a different context.

SF_NOCACHE — The data sent to socket will not be cached by the virtual memory system, and will be freed directly to the pool of free pages.

SF_SYNC — **sendfile** sleeps until the network stack no longer references the VM pages of the file, making subsequent modifications to it safe. Please note that this is not a guarantee that the data has actually been sent.

SF_USER_READAHEAD — **sendfile** has some internal heuristics to do readahead when sending data. This flag forces **sendfile** to override any heuristically calculated readahead and use exactly the application specified readahead. See *SETTING READAHEAD* for more details on readahead.

When using a socket marked for non-blocking I/O, **sendfile**() may send fewer bytes than requested. In this case, the number of bytes successfully written is returned in **sbytes* (if specified), and the error EAGAIN is returned.

SETTING READAHEAD

sendfile uses internal heuristics based on request size and file system layout to do readahead. Additionally application may request extra readahead. The most significant 16 bits of *flags* specify amount of pages that **sendfile** may read ahead when reading the file. A macro **SF_FLAGS**() is provided to combine readahead amount and flags. An example showing specifying readahead of 16 pages and SF_NOCACHE flag:

SF_FLAGS(16, SF_NOCACHE)

sendfile will use either application specified readahead or internally calculated, whichever is bigger. Setting flag SF_USER_READAHEAD would turn off any heuristics and set maximum possible readahead length to the number of pages specified via flags.

IMPLEMENTATION NOTES

The FreeBSD implementation of **sendfile**() does not block on disk I/O when it sends a file off the ffs(7) filesystem. The syscall returns success before the actual I/O completes, and data is put into the socket later unattended. However, the order of data in the socket is preserved, so it is safe to do further writes to the socket.

The FreeBSD implementation of **sendfile**() is "zero-copy", meaning that it has been optimized so that copying of the file data is avoided.

TUNING

physical paging buffers

sendfile() uses vnode pager to read file pages into memory. The pager uses a pool of physical buffers to run its I/O operations. When system runs out of pbufs, sendfile will block and report state "zonelimit". Size of the pool can be tuned with *vm.vnode_pbufs* loader.conf(5) tunable and can be checked with sysctl(8) OID of the same name at runtime.

sendfile(2) buffers

On some architectures, this system call internally uses a special **sendfile**() buffer (*struct sf_buf*) to handle sending file data to the client. If the sending socket is blocking, and there are not enough **sendfile**() buffers available, **sendfile**() will block and report a state of "sfbufa". If the sending socket is non-blocking and there are not enough **sendfile**() buffers available, the call will block and wait for the necessary buffers to become available before finishing the call.

The number of *sf_buf*'s allocated should be proportional to the number of nmbclusters used to send data to a client via **sendfile**(). Tune accordingly to avoid blocking! Busy installations that make extensive use of **sendfile**() may want to increase these values to be inline with their *kern.ipc.nmbclusters* (see tuning(7) for details).

The number of **sendfile**() buffers available is determined at boot time by either the *kern.ipc.nsfbufs* loader.conf(5) variable or the NSFBUFS kernel configuration tunable. The number of **sendfile**() buffers scales with *kern.maxusers*. The *kern.ipc.nsfbufsused* and *kern.ipc.nsfbufspeak* read-only sysctl(8) variables show current and peak **sendfile**() buffers usage respectively. These values may also be viewed through **netstat -m**.

If sysctl(8) OID *kern.ipc.nsfbufs* doesn't exist, your architecture does not need to use **sendfile**() buffers because their task can be efficiently performed by the generic virtual memory structures.

RETURN VALUES

The **sendfile**() function returns the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

[EAGAIN] The socket is marked for non-blocking I/O and not all data was sent due to the socket buffer being filled. If specified, the number of bytes successfully sent will be returned in **sbytes*.

[EBADF] The *fd* argument is not a valid file descriptor.

[EBADF] The *s* argument is not a valid socket descriptor.

[EBUSY] A busy page was encountered and SF_NODISKIO had been specified. Partial data may have been sent.

[EFAULT] An invalid address was specified for an argument.

[EINTR] A signal interrupted **sendfile**() before it could be completed. If specified, the number of bytes successfully sent will be returned in **sbytes*.

[EINVAL] The *fd* argument is not a regular file.

[EINVAL] The *s* argument is not a SOCK_STREAM type socket.

[EINVAL] The *offset* argument is negative.

[EIO] An error occurred while reading from *fd*.

[EINTEGRITY] Corrupted data was detected while reading from *fd*.

[ENOTCAPABLE] The *fd* or the *s* argument has insufficient rights.

[ENOBUFS] The system was unable to allocate an internal buffer.

[ENOTCONN] The *s* argument points to an unconnected socket.

[ENOTSOCK] The *s* argument is not a socket.

[EOPNOTSUPP] The file system for descriptor *fd* does not support **sendfile**().

[EPIPE] The socket peer has closed the connection.

SEE ALSO

netstat(1), open(2), send(2), socket(2), writev(2), loader.conf(5), tuning(7), sysctl(8)

K. Elmeleegy, A. Chanda, A. L. Cox, and W. Zwaenepoel, "A Portable Kernel Abstraction for Low-Overhead Ephemeral Mapping Management", *The Proceedings of the 2005 USENIX Annual Technical Conference*, pp 223-236, 2005.

HISTORY

The **sendfile**() system call first appeared in FreeBSD 3.0. This manual page first appeared in FreeBSD 3.1. In FreeBSD 10 support for sending shared memory descriptors had been introduced. In FreeBSD 11 a non-blocking implementation had been introduced.

AUTHORS

The initial implementation of **sendfile**() system call and this manual page were written by David G. Lawrence <*dg@dglawrence.com*>. The FreeBSD 11 implementation was written by Gleb Smirnoff <*glebius@FreeBSD.org*>.

BUGS

The **sendfile**() system call will not fail, i.e., return -1 and set *errno* to EFAULT, if provided an invalid address for *sbytes*. The **sendfile**() system call does not support SCTP sockets, it will return -1 and set *errno* to EINVAL.