



*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 'sendfile.2' command**

### **\$ man sendfile.2**

SENDFILE(2)            Linux Programmer's Manual            SENDFILE(2)

#### NAME

sendfile - transfer data between file descriptors

#### SYNOPSIS

```
#include <sys/sendfile.h>

ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

#### DESCRIPTION

sendfile() copies data between one file descriptor and another. Because this copying is done within the kernel, sendfile() is more efficient than the combination of read(2) and write(2), which would require transferring data to and from user space.

in\_fd should be a file descriptor opened for reading and out\_fd should be a descriptor opened for writing.

If offset is not NULL, then it points to a variable holding the file offset from which sendfile() will start reading data from in\_fd. When sendfile() returns, this variable will be set to the offset of the byte following the last byte that was read. If offset is not NULL, then sendfile() does not modify the file offset of in\_fd; otherwise the file offset is adjusted to reflect the number of bytes read from in\_fd.

If offset is NULL, then data will be read from in\_fd starting at the file offset, and the file offset will be updated by the call.

count is the number of bytes to copy between the file descriptors.

The in\_fd argument must correspond to a file which supports

mmap(2)-like operations (i.e., it cannot be a socket).

In Linux kernels before 2.6.33, `out_fd` must refer to a socket. Since Linux 2.6.33 it can be any file. If it is a regular file, then `sendfile()` changes the file offset appropriately.

## RETURN VALUE

If the transfer was successful, the number of bytes written to `out_fd` is returned. Note that a successful call to `sendfile()` may write fewer bytes than requested; the caller should be prepared to retry the call if there were unsent bytes. See also NOTES.

On error, -1 is returned, and `errno` is set appropriately.

## ERRORS

**EAGAIN** Nonblocking I/O has been selected using `O_NONBLOCK` and the write would block.

**EBADF** The input file was not opened for reading or the output file was not opened for writing.

**EFAULT** Bad address.

**EINVAL** Descriptor is not valid or locked, or an mmap(2)-like operation is not available for `in_fd`, or `count` is negative.

**EINVAL** `out_fd` has the `O_APPEND` flag set. This is not currently supported by `sendfile()`.

**EIO** Unspecified error while reading from `in_fd`.

**ENOMEM** Insufficient memory to read from `in_fd`.

### **EOVERFLOW**

`count` is too large, the operation would result in exceeding the maximum size of either the input file or the output file.

**ESPIPE** `offset` is not NULL but the input file is not seekable.

## VERSIONS

`sendfile()` first appeared in Linux 2.2. The include file `<sys/sendfile.h>` is present since glibc 2.1.

## CONFORMING TO

Not specified in POSIX.1-2001, nor in other standards.

Other UNIX systems implement `sendfile()` with different semantics and prototypes. It should not be used in portable programs.

## NOTES

`sendfile()` will transfer at most 0x7ffff000 (2,147,479,552) bytes, returning the number of bytes actually transferred. (This is true on both 32-bit and 64-bit systems.)

If you plan to use `sendfile()` for sending files to a TCP socket, but need to send some header data in front of the file contents, you will find it useful to employ the `TCP_CORK` option, described in `tcp(7)`, to minimize the number of packets and to tune performance.

In Linux 2.4 and earlier, `out_fd` could also refer to a regular file; this possibility went away in the Linux 2.6.x kernel series, but was restored in Linux 2.6.33.

The original Linux `sendfile()` system call was not designed to handle large file offsets. Consequently, Linux 2.4 added `sendfile64()`, with a wider type for the offset argument. The glibc `sendfile()` wrapper function transparently deals with the kernel differences.

Applications may wish to fall back to `read(2)/write(2)` in the case where `sendfile()` fails with `EINVAL` or `ENOSYS`.

If `out_fd` refers to a socket or pipe with zero-copy support, callers must ensure the transferred portions of the file referred to by `in_fd` remain unmodified until the reader on the other end of `out_fd` has consumed the transferred data.

The Linux-specific `splice(2)` call supports transferring data between arbitrary file descriptors provided one (or both) of them is a pipe.

## SEE ALSO

`copy_file_range(2)`, `mmap(2)`, `open(2)`, `socket(2)`, `splice(2)`

## COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.