



Red Hat Enterprise Linux Release 9.2 Manual Pages on 'send.2' command

\$ man send.2

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

NAME

send, sendto, sendmsg - send a message on a socket

SYNOPSIS

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

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
```

DESCRIPTION

The system calls send(), sendto(), and sendmsg() are used to transmit a message to another socket.

The send() call may be used only when the socket is in a connected state (so that the intended recipient is known). The only difference between send() and write(2) is the presence of flags. With a zero flags argument, send() is equivalent to write(2). Also, the following call

```
send(sockfd, buf, len, flags);
```

is equivalent to

```
sendto(sockfd, buf, len, flags, NULL, 0);
```

The argument sockfd is the file descriptor of the sending socket.

If sendto() is used on a connection-mode (SOCK_STREAM, SOCK_SEQPACKET)

socket, the arguments `dest_addr` and `addrlen` are ignored (and the error `EISCONN` may be returned when they are not `NULL` and `0`), and the error `ENOTCONN` is returned when the socket was not actually connected. Otherwise, the address of the target is given by `dest_addr` with `addrlen` specifying its size. For `sendmsg()`, the address of the target is given by `msg.msg_name`, with `msg.msg_namelen` specifying its size.

For `send()` and `sendto()`, the message is found in `buf` and has length `len`. For `sendmsg()`, the message is pointed to by the elements of the array `msg.msg_iov`. The `sendmsg()` call also allows sending ancillary data (also known as control information).

If the message is too long to pass atomically through the underlying protocol, the error `EMSGSIZE` is returned, and the message is not transmitted.

No indication of failure to deliver is implicit in a `send()`. Locally detected errors are indicated by a return value of `-1`.

When the message does not fit into the send buffer of the socket, `send()` normally blocks, unless the socket has been placed in nonblocking I/O mode. In nonblocking mode it would fail with the error `EAGAIN` or `EWOULDBLOCK` in this case. The `select(2)` call may be used to determine when it is possible to send more data.

The flags argument

The flags argument is the bitwise OR of zero or more of the following flags.

`MSG_CONFIRM` (since Linux 2.3.15)

Tell the link layer that forward progress happened: you got a successful reply from the other side. If the link layer doesn't get this it will regularly reprobe the neighbor (e.g., via a unicast ARP). Valid only on `SOCK_DGRAM` and `SOCK_RAW` sockets and currently implemented only for IPv4 and IPv6. See `arp(7)` for details.

`MSG_DONTROUTE`

Don't use a gateway to send out the packet, send to hosts only on directly connected networks. This is usually used only by

diagnostic or routing programs. This is defined only for proto?

col families that route; packet sockets don't.

MSG_DONTWAIT (since Linux 2.2)

Enables nonblocking operation; if the operation would block, EA?

GAIN or EWOULDBLOCK is returned. This provides similar behavior

to setting the O_NONBLOCK flag (via the fcntl(2) F_SETFL opera?

tion), but differs in that MSG_DONTWAIT is a per-call option,

whereas O_NONBLOCK is a setting on the open file description

(see open(2)), which will affect all threads in the calling

process and as well as other processes that hold file descrip?

tors referring to the same open file description.

MSG_EOR (since Linux 2.2)

Terminates a record (when this notion is supported, as for sock?

ets of type SOCK_SEQPACKET).

MSG_MORE (since Linux 2.4.4)

The caller has more data to send. This flag is used with TCP

sockets to obtain the same effect as the TCP_CORK socket option

(see tcp(7)), with the difference that this flag can be set on a

per-call basis.

Since Linux 2.6, this flag is also supported for UDP sockets,

and informs the kernel to package all of the data sent in calls

with this flag set into a single datagram which is transmitted

only when a call is performed that does not specify this flag.

(See also the UDP_CORK socket option described in udp(7).)

MSG_NOSIGNAL (since Linux 2.2)

Don't generate a SIGPIPE signal if the peer on a stream-oriented

socket has closed the connection. The EPIPE error is still re?

turned. This provides similar behavior to using sigaction(2) to

ignore SIGPIPE, but, whereas MSG_NOSIGNAL is a per-call feature,

ignoring SIGPIPE sets a process attribute that affects all

threads in the process.

MSG_OOB

Sends out-of-band data on sockets that support this notion

(e.g., of type SOCK_STREAM); the underlying protocol must also support out-of-band data.

sendmsg()

The definition of the msghdr structure employed by sendmsg() is as follows:

```
struct msghdr {  
    void      *msg_name;    /* Optional address */  
    socklen_t  msg_namelen; /* Size of address */  
    struct iovec *msg_iov;   /* Scatter/gather array */  
    size_t     msg_iovlen;  /* # elements in msg_iov */  
    void      *msg_control; /* Ancillary data, see below */  
    size_t     msg_controllen; /* Ancillary data buffer len */  
    int        msg_flags;    /* Flags (unused) */  
};
```

The msg_name field is used on an unconnected socket to specify the target address for a datagram. It points to a buffer containing the address; the msg_namelen field should be set to the size of the address. For a connected socket, these fields should be specified as NULL and 0, respectively.

The msg_iov and msg_iovlen fields specify scatter-gather locations, as for writev(2).

You may send control information (ancillary data) using the msg_control and msg_controllen members. The maximum control buffer length the kernel can process is limited per socket by the value in /proc/sys/net/core/optmem_max; see socket(7). For further information on the use of ancillary data in various socket domains, see unix(7) and ip(7).

The msg_flags field is ignored.

RETURN VALUE

On success, these calls return the number of bytes sent. On error, -1 is returned, and errno is set appropriately.

ERRORS

These are some standard errors generated by the socket layer. Addi?

tional errors may be generated and returned from the underlying proto?
col modules; see their respective manual pages.

EACCES (For UNIX domain sockets, which are identified by pathname)

Write permission is denied on the destination socket file, or
search permission is denied for one of the directories the path
prefix. (See path_resolution(7).)

(For UDP sockets) An attempt was made to send to a net?
work/broadcast address as though it was a unicast address.

EAGAIN or EWOULDBLOCK

The socket is marked nonblocking and the requested operation
would block. POSIX.1-2001 allows either error to be returned
for this case, and does not require these constants to have the
same value, so a portable application should check for both pos?
sibilities.

EAGAIN (Internet domain datagram sockets) The socket referred to by

sockfd had not previously been bound to an address and, upon at?
tempting to bind it to an ephemeral port, it was determined that
all port numbers in the ephemeral port range are currently in
use. See the discussion of /proc/sys/net/ipv4/ip_lo?
cal_port_range in ip(7).

EALREADY

Another Fast Open is in progress.

EBADF sockfd is not a valid open file descriptor.

ECONNRESET

Connection reset by peer.

EDESTADDRREQ

The socket is not connection-mode, and no peer address is set.

EFAULT An invalid user space address was specified for an argument.

EINTR A signal occurred before any data was transmitted; see sig?
nal(7).

EINVAL Invalid argument passed.

EISCONN

The connection-mode socket was connected already but a recipient

was specified. (Now either this error is returned, or the recipient specification is ignored.)

EMSGSIZE

The `socket` type requires that message be sent atomically, and the size of the message to be sent made this impossible.

ENOBUFS

The output queue for a network interface was full. This generally indicates that the interface has stopped sending, but may be caused by transient congestion. (Normally, this does not occur in Linux. Packets are just silently dropped when a device queue overflows.)

ENOMEM No memory available.

ENOTCONN

The socket is not connected, and no target has been given.

ENOTSOCK

The file descriptor `sockfd` does not refer to a socket.

EOPNOTSUPP

Some bit in the `flags` argument is inappropriate for the socket type.

EPIPE The local end has been shut down on a connection oriented socket. In this case, the process will also receive a `SIGPIPE` unless `MSG_NOSIGNAL` is set.

CONFORMING TO

4.4BSD, SVr4, POSIX.1-2001. These interfaces first appeared in 4.2BSD.

POSIX.1-2001 describes only the `MSG_OOB` and `MSG_EOR` flags.

POSIX.1-2008 adds a specification of `MSG_NOSIGNAL`. The `MSG_CONFIRM` flag is a Linux extension.

NOTES

According to POSIX.1-2001, the `msg_controllen` field of the `msghdr` structure should be typed as `socklen_t`, and the `msg_iovlen` field should be typed as `int`, but glibc currently types both as `size_t`.

See `sendmmsg(2)` for information about a Linux-specific system call that can be used to transmit multiple datagrams in a single call.

BUGS

Linux may return EPIPE instead of ENOTCONN.

EXAMPLES

An example of the use of `sendto()` is shown in `getaddrinfo(3)`.

SEE ALSO

`fcntl(2)`, `getsockopt(2)`, `recv(2)`, `select(2)`, `sendfile(2)`, `sendmmsg(2)`,
`shutdown(2)`, `socket(2)`, `write(2)`, `cmsg(3)`, `ip(7)`, `ipv6(7)`, `socket(7)`,
`tcp(7)`, `udp(7)`, `unix(7)`

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/>.

Linux

2020-11-01

SEND(2)