sendto(2) - Linux man page

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 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). Only valid on **SOCK_DGRAM** and **SOCK_RAW** sockets and currently only implemented for IPv4 and IPv6. See **arp**(7) for details.

MSG_DONTROUTE
: Don't use a gateway to send out the packet, only send to hosts on directly connected networks. This is usually used only by diagnostic or routing programs. This is only defined for protocol families that route; packet sockets don't.

MSG_DONTWAIT (since Linux 2.2)
: Enables nonblocking operation; if the operation would block, **EAGAIN** or **EWOULDBLOCK** is returned (this can also be enabled using the **O_NONBLOCK** flag with the **F_SETFL fcntl**(2)).

MSG_EOR (since Linux 2.2)
: Terminates a record (when this notion is supported, as for sockets 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 only transmitted 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)
Requests not to send **SIGPIPE** on errors on stream oriented sockets when the other end breaks the connection. The **EPIPE** error is still returned.

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.

The definition of the *msghdr* structure follows. See **recv**(2) and below for an exact description of its fields.

```
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 on received message */
};
```

You may send control information 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).

Return Value

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

Errors

These are some standard errors generated by the socket layer. Additional errors may be generated and returned from the underlying protocol 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 network/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 possibilities.

EBADF

An invalid descriptor was specified.

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 **signal**(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 argument *sockfd* is not 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 function calls appeared in 4.2BSD.

POSIX.1-2001 only describes 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

The prototypes given above follow the Single UNIX Specification, as glibc2 also does; the *flags* argument was *int* in 4.x BSD, but *unsigned int* in libc4 and libc5; the *len* argument was *int* in 4.x BSD and libc4, but *size_t* in libc5; the *addrlen* argument was *int* in 4.x BSD and libc4 and libc5. See also **accept**(2).

According to POSIX.1-2001, the *msg_controllen* field of the *msghdr* structure should be typed as *socklen_t*, but glibc currently types it 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**.

Example

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), **socket**(7), **tcp**(7), **udp**(7)

Referenced By

getifaddrs(3), **if_nameindex**(3), **lwres_getaddrinfo**(3), **pth**(3), **rtime**(3), **socketcall**(2)