## NAME

**sctp_send**, **sctp_sendx** - send a message from an SCTP socket

## LIBRARY

Standard C Library (libc, -lc)

## SYNOPSIS

**#include <sys/types.h>**
**#include <sys/socket.h>**
**#include <netinet/sctp.h>**

*ssize_t*
**sctp_send**(*int sd*, *const void *msg*, *size_t len*, *const struct sctp_sndrcvinfo *sinfo*, *int flags*);

*ssize_t*
**sctp_sendx**(*int sd*, *const void *msg*, *size_t len*, *struct sockaddr *addrs*, *int addrcnt*,
    *const struct sctp_sndrcvinfo *sinfo*, *int flags*);

## DESCRIPTION

The **sctp_send**() system call is used to transmit a message to another SCTP endpoint. **sctp_send**() may be used to send data to an existing association for both one-to-many (SOCK_SEQPACKET) and one-to-one (SOCK_STREAM) socket types. The length of the message *msg* is given by *len*. If the message is too long to pass atomically through the underlying protocol, *errno* is set to EMSGSIZE, -1 is returned, and the message is not transmitted.

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

If no space is available at the socket to hold the message to be transmitted, then **sctp_send**() normally blocks, unless the socket has been placed in non-blocking I/O mode. The select(2) system call may be used to determine when it is possible to send more data on one-to-one type (SOCK_STREAM) sockets.

The *sinfo* structure is used to control various SCTP features and has the following format:

```
struct sctp_sndrcvinfo {
        uint16_t sinfo_stream;  /* Stream sending to */
        uint16_t sinfo_ssn;     /* valid for recv only */
        uint16_t sinfo_flags;   /* flags to control sending */
        uint32_t sinfo_ppid;    /* ppid field */
        uint32_t sinfo_context; /* context field */
        uint32_t sinfo_timetolive; /* timetolive for PR-SCTP */
        uint32_t sinfo_tsn;        /* valid for recv only */
        uint32_t sinfo_cumtsn;     /* valid for recv only */
        sctp_assoc_t sinfo_assoc_id; /* The association id */
};
```

The *sinfo->sinfo_ppid* argument is an opaque 32 bit value that is passed transparently through the stack to the peer endpoint. It will be available on reception of a message (see sctp_recvmsg(3)). Note that the stack passes this value without regard to byte order.

The *sinfo->sinfo_flags* argument may include one or more of the following:

```
#define SCTP_EOF              0x0100  /* Start a shutdown procedures */
#define SCTP_ABORT            0x0200  /* Send an ABORT to peer */
#define SCTP_UNORDERED                0x0400  /* Message is un-ordered */
#define SCTP_ADDR_OVER                0x0800  /* Override the primary-address */
#define SCTP_SENDALL      0x1000   /* Send this on all associations */
                                          /* for the endpoint */
/* The lower byte is an enumeration of PR-SCTP policies */
#define SCTP_PR_SCTP_TTL  0x0001          /* Time based PR-SCTP */
#define SCTP_PR_SCTP_BUF  0x0002          /* Buffer based PR-SCTP */
#define SCTP_PR_SCTP_RTX  0x0003          /* Number of retransmissions based PR-SCTP */
```

The flag SCTP_EOF is used to instruct the SCTP stack to queue this message and then start a graceful shutdown of the association. All remaining data in queue will be sent after which the association will be shut down.

SCTP_ABORT is used to immediately terminate an association. An abort is sent to the peer and the local TCB is destroyed.

SCTP_UNORDERED is used to specify that the message being sent has no specific order and should be delivered to the peer application as soon as possible. When this flag is absent messages are delivered in order within the stream they are sent, but without respect to order to peer streams.

The flag SCTP_ADDR_OVER is used to specify that a specific address should be used. Normally SCTP will use only one of a multi-homed peers addresses as the primary address to send to. By default, no matter what the *to* argument is, this primary address is used to send data. By specifying this flag, the user is asking the stack to ignore the primary address and instead use the specified address not only as a lookup mechanism to find the association but also as the actual address to send to.

For a one-to-many type (SOCK_SEQPACKET) socket the flag SCTP_SENDALL can be used as a convenient way to make one send call and have all associations that are under the socket get a copy of the message. Note that this mechanism is quite efficient and makes only one actual copy of the data which is shared by all the associations for sending.

The remaining flags are used for the partial reliability extension (RFC3758) and will only be effective if the peer endpoint supports this extension. This option specifies what local policy the local endpoint should use in skipping data. If none of these options are set, then data is never skipped over.

SCTP_PR_SCTP_TTL is used to indicate that a time based lifetime is being applied to the data. The *sinfo->sinfo_timetolive* argument is then a number of milliseconds for which the data is attempted to be transmitted. If that many milliseconds elapse and the peer has not acknowledged the data, the data will be skipped and no longer transmitted. Note that this policy does not even assure that the data will ever be sent. In times of a congestion with large amounts of data being queued, the *sinfo->sinfo_timetolive* may expire before the first transmission is ever made.

The SCTP_PR_SCTP_BUF based policy transforms the *sinfo->sinfo_timetolive* field into a total number of bytes allowed on the outbound send queue. If that number or more bytes are in queue, then other buffer-based sends are looked to be removed and skipped. Note that this policy may also result in the data never being sent if no buffer based sends are in queue and the maximum specified by *timetolive* bytes is in queue.

The SCTP_PR_SCTP_RTX policy transforms the *sinfo->sinfo_timetolive* into a number of retransmissions to allow. This policy always assures that at a minimum one send attempt is made of the data. After which no more than *sinfo->sinfo_timetolive* retransmissions will be made before the data is skipped.

*sinfo->sinfo_stream* is the SCTP stream that you wish to send the message on. Streams in SCTP are reliable (or partially reliable) flows of ordered messages.

The *sinfo->sinfo_assoc_id* field is used to select the association to send to on a one-to-many socket. For a one-to-one socket, this field is ignored.

The *sinfo->sinfo_context* field is used only in the event the message cannot be sent. This is an opaque value that the stack retains and will give to the user when a failed send is given if that notification is enabled (see sctp(4)). Normally a user process can use this value to index some application specific data structure when a send cannot be fulfilled.

The *flags* argument holds the same meaning and values as those found in sendmsg(2) but is generally ignored by SCTP.

The fields *sinfo->sinfo_ssn*, *sinfo->sinfo_tsn*, and *sinfo->sinfo_cumtsn* are used only when receiving messages and are thus ignored by **sctp_send**(). The function **sctp_sendx**() has the same properties as **sctp_send**() with the additional arguments of an array of sockaddr structures passed in. With the *addrs* argument being given as an array of addresses to be sent to and the *addrcnt* argument indicating how many socket addresses are in the passed in array. Note that all of the addresses will only be used when an implicit association is being set up. This allows the user the equivalent behavior as doing a **sctp_connectx**() followed by a **sctp_send**() to the association. Note that if the *sinfo->sinfo_assoc_id* field is 0, then the first address will be used to look up the association in place of the association id. If both an address and an association id are specified, the association id has priority.

## RETURN VALUES

The call returns the number of characters sent, or -1 if an error occurred.

## ERRORS

The **sctp_send**() system call fails if:

| | |
|---|---|
| [EBADF] | An invalid descriptor was specified. |
| [ENOTSOCK] | The argument *s* is not a socket. |
| [EFAULT] | An invalid user space address was specified for an argument. |
| [EMSGSIZE] | The socket requires that message be sent atomically, and the size of the message to be sent made this impossible. |
| [EAGAIN] | The socket is marked non-blocking and the requested operation would block. |
| [ENOBUFS] | The system was unable to allocate an internal buffer. The operation may succeed when buffers become available. |
| [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. |
| [EHOSTUNREACH] | The remote host was unreachable. |
| [ENOTCONN] | On a one-to-one style socket no association exists. |
| [ECONNRESET] | An abort was received by the stack while the user was attempting to send data to the peer. |

| | |
|---|---|
| [ENOENT] | On a one-to-many style socket no address is specified so that the association cannot be located or the SCTP_ABORT flag was specified on a non-existing association. |
| [EPIPE] | The socket is unable to send anymore data (SBS_CANTSENDMORE has been set on the socket). This typically means that the socket is not connected and is a one-to-one style socket. |

## SEE ALSO

getsockopt(2), recv(2), select(2), sendmsg(2), socket(2), write(2), sctp_connectx(3), sctp_recvmsg(3), sctp_sendmsg(3), sctp(4)

## BUGS

Because **sctp_send**() may have multiple associations under one endpoint, a select on write will only work for a one-to-one style socket.