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

NAME

recv, recvfrom, recvmsg - receive a message from a socket

SYNOPSIS

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

int recv(int s, void *buf, int len, unsigned int flags);

int recvfrom(int s, void *buf, int len, unsigned int flags
struct sockaddr *from, int *fromlen);

int recvmsg(int s, struct msghdr *msg, unsigned int
flags);
```

DESCRIPTION

The **recvfrom** and **recvmsg** calls are used to receive messages from a socket, and may be used to receive data on a socket whether or not it is connection-oriented.

If *from* is not NULL, and the socket is not connection-oriented, the source address of the message is filled in. *Fromlen* is a value-result parameter, initialized to the size of the buffer associated with *from*, and modified on return to indicate the actual size of the address stored there.

The **recv** call is normally used only on a *connected* socket (see **connect**(2)) and is identical to **recvfrom** with a NULL *from* parameter. As it is redundant, it may not be supported in future releases.

All three routines return the length of the message on successful completion. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see **socket**(2)).

If no messages are available at the socket, the receive calls wait for a message to arrive, unless the socket is nonblocking (see **fcntl**(2)) in which case the value -1 is returned and the external variable *errno* set to **EAGAIN**. The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested.

The **select**(2) or **poll**(2) call may be used to determine when more data arrives.

The *flags* argument to a recv call is formed by *OR*'ing one or more of the following values:

MSG_OOB
This flag requests receipt of out-of-band data that would not be received in the normal data stream. Some protocols place expedited data at the head of the normal data queue, and thus this flag cannot be used with such protocols.

MSG_PEEK
This flag causes the receive operation to return data from the beginning of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same data.

MSG_WAITALL
This flag requests that the operation block until the full request is satisfied. However, the call may still return less data than requested if a signal is caught, an error or disconnect occurs, or the next data to be received is of a different type than that returned.

MSG_ERRQUEUE
Receive packet from the error queue

MSG_NOSIGNAL
This flag turns off raising of **SIGPIPE on stream sockets when the** other end disappears.

MSG_ERRQUEUE
This flag specifies that queued errors should be received from the socket error queue. The error is passed in a ancilliary message with a type dependent on the protocol (for IP **IP_RECVERR**). The error is supplied in a **sock_extended_error** structure:

```
#define SO_EE_ORIGIN_NONE       0
#define SO_EE_ORIGIN_LOCAL      1
#define SO_EE_ORIGIN_ICMP       2
#define SO_EE_ORIGIN_ICMP6      3

struct sock_extended_err
{
    __u32           ee_errno;   /* error number */
    __u8            ee_origin;  /* where the error originated */
    __u8            ee_type;    /* type */
    __u8            ee_code;    /* code */
    __u8            ee_pad;
    __u32           ee_info;    /* additional information */
    __u32           ee_data;    /* other data */
};

struct sockaddr *SOCK_EE_OFFENDER(struct sock_extended_err *);
```

ee_errno contains the errno number of the queued

error. **ee_origin** is the origin code of where the error originated. The other fields are protocol specific. **SOCK_EE_OFFENDER** returns a pointer to the address of the network object where the error originated from. If this address is not known, the *sa_family* member of the **sockaddr** contains **AF_UNSPEC** and the other fields of the **sockaddr** are undefined. The payload of the packet that caused the error is passed as normal data.

For local errors, no address is passed (this can be checked with the *cmsg_len* member of the **cmsghdr**). For error receives, the **MSG_ERRQUEUE** is set in the **msghdr**. After a error has been passed, the pending socket error is regenerated based on the next queued error and will be passed on the next socket operation.

The **recvmsg** call uses a **msghdr** structure to minimize the number of directly supplied parameters. This structure has the following form, as defined in **<sys/socket.h>**:

```
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 */
    socklen_t   msg_controllen; /* ancillary data buffer len */
    int         msg_flags;      /* flags on received message */
};
```

Msg_name and *msg_namelen* specify the destination address if the socket is unconnected; *msg_name* may be given as a null pointer if no names are desired or required. *Msg_iov* and *msg_iovlen* describe scatter-gather locations, as discussed in **readv**(2). *msg_control*, which has length *msg_controllen*, points to a buffer for other protocol control related messages or miscellaneous ancillary data. When **recvmsg** is called, *msg_controllen* should contain the length of the available buffer in *msg_control*; after the successful call return it will contain the length of the control message sequence.

The messages are of the form:

```
struct cmsghdr {
    socklen_t   cmsg_len;       /* data byte count, including hdr */
    int         cmsg_level;     /* originating protocol */
    int         cmsg_type;      /* protocol-specific type */
/* followed by
    u_char      cmsg_data[]; */
};
```

Ancillary data should only be accessed by the macros

defined in **cmsg**(3).

As an example, Linux uses this auxiliary data mechanism to pass extended errors, IP options or file descriptors over Unix sockets.

The *msg_flags* field is set on return according to the message received. **MSG_EOR** indicates end-of-record; the data returned completed a record (generally used with sockets of type **SOCK_SEQPACKET**). **MSG_TRUNC** indicates that the trailing portion of a datagram was discarded because the datagram was larger than the buffer supplied. **MSG_CTRUNC** indicates that some control data were discarded due to lack of space in the buffer for ancillary data. **MSG_OOB** is returned to indicate that expedited or out-of-band data were received. **MSG_ERRQUEUE** indicates that no data was received but an extended error from the socket error queue.

RETURN VALUES

These calls return the number of bytes received, or -1 if an error occurred.

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 manual pages.

EBADF The argument *s* is an invalid descriptor.

ENOTCONN
The socket is associated with a connection-oriented protocol and has not been connected (see **connect**(2) and **accept**(2)).

ENOTSOCK
The argument *s* does not refer to a socket.

EAGAIN The socket is marked non-blocking and the receive operation would block, or a receive timeout had been set and the timeout expired before data was received.

EINTR The receive was interrupted by delivery of a signal before any data were available.

EFAULT The receive buffer pointer(s) point outside the process's address space.

EINVAL Invalid argument passed.

CONFORMING TO

4.4BSD (these function calls first appeared in 4.2BSD).

SEE ALSO

fcntl(2), read(2), select(2), getsockopt(2), socket(2), cmsg(3)

BSD Man Page 24 July 1993 1