NAME

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

SYNOPSIS

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

ssize_t recv(int *sockfd***, void ****buf***, size_t** *len***, int** *flags***);**

ssize_t recvfrom(int *sockfd***, void ****buf***, size_t** *len***, int** *flags***,**
struct sockaddr ****src_addr*, socklen_t ****addrlen***);**

ssize_t recvmsg(int *sockfd***, struct msghdr ****msg***, int** *flags***);**

DESCRIPTION

The **recv**(), **recvfrom**(), and **recvmsg**() calls are used to receive messages from a socket. They may be used to receive data on both connectionless and connection-oriented sockets. This page first describes common features of all three system calls, and then describes the differences between the calls.

The only difference between **recv**() and read(2) is the presence of *flags*. With a zero *flags* argument, **recv**() is generally equivalent to read(2) (but see NOTES). Also, the following call

recv(sockfd, buf, len, flags);

is equivalent to

recvfrom(sockfd, buf, len, flags, NULL, NULL);

All three calls 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.

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* is set to **EAGAIN** or **EWOULDBLOCK**. The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested.

An application can use select(2), poll(2), or epoll(7) to determine when more data arrives on a socket.

The flags argument

The *flags* argument is formed by ORing one or more of the following values:

MSG_CMSG_CLOEXEC (**recvmsg**() only; since Linux 2.6.23)
Set the close-on-exec flag for the file descriptor received via a UNIX domain file descriptor using the **SCM_RIGHTS** operation (described in unix(7)). This flag is useful for the same reasons as the **O_CLOEXEC** flag of open(2).

MSG_DONTWAIT (since Linux 2.2)
Enables nonblocking operation; if the operation would block, the call fails with the error **EAGAIN** or **EWOULDBLOCK**. This provides similar behavior to setting the **O_NONBLOCK** flag (via the fcntl(2) **F_SETFL** operation), 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 descriptors referring to the same open file description.

MSG_ERRQUEUE (since Linux 2.2)
This flag specifies that queued errors should be received from the socket error queue. The error is passed in an ancillary message with a type dependent on the protocol (for IPv4 **IP_RECVERR**). The user should supply a buffer of sufficient size. See cmsg(3) and ip(7) for more information. The payload of the original packet that caused the error is passed as normal data via *msg_iovec*. The original destination address of the datagram that caused the error is supplied via *msg_name*.

The error is supplied in a *sock_extended_err* 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
{
uint32_t ee_errno;   /* error number */
uint8_t  ee_origin;  /* where the error originated */
uint8_t  ee_type;    /* type */
uint8_t  ee_code;    /* code */
uint8_t  ee_pad;     /* padding */
uint32_t ee_info;    /* additional information */
uint32_t ee_data;    /* other data */
/* More data may follow */
};

struct sockaddr *SO_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. The macro **SOCK_EE_OFFENDER** returns a pointer to the address of the network object where the error originated from given a pointer to the ancillary message. 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** flag is set in the *msghdr*. After an 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.

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_TRUNC (since Linux 2.2)
For raw (**AF_PACKET**), Internet datagram (since Linux 2.4.27/2.6.8), netlink (since Linux 2.6.22), and UNIX datagram (since Linux 3.4) sockets: return the real length of the packet or datagram, even when it was longer than the passed buffer.

For use with Internet stream sockets, see tcp(7).

MSG_WAITALL (since Linux 2.2)
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. This flag has no effect for datagram sockets.

recvfrom()

recvfrom() places the received message into the buffer *buf*. The caller must specify the size of the buffer in *len*.

If *src_addr* is not NULL, and the underlying protocol provides the source address of the message, that source address is placed in the buffer pointed to by *src_addr*. In this case, *addrlen* is a value-result argument. Before the call, it should be initialized to the size of the buffer associated with *src_addr*. Upon return, *addrlen* is updated to contain the actual size of the source address. The returned address is truncated

if the buffer provided is too small; in this case, *addrlen* will return a value greater than was supplied to the call.

If the caller is not interested in the source address, *src_addr* and *addrlen* should be specified as NULL.

recv()

The **recv**() call is normally used only on a *connected* socket (see connect(2)). It is equivalent to the call:

recvfrom(fd, buf, len, flags, NULL, 0);

recvmsg()

The **recvmsg**() call uses a *msghdr* structure to minimize the number of directly supplied arguments. This structure is defined as follows in *<sys/socket.h>*:

```
struct iovec {                    /* Scatter/gather array items */
void  *iov_base;              /* Starting address */
size_t iov_len;               /* Number of bytes to transfer */
};

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 */
};
```

The *msg_name* field points to a caller-allocated buffer that is used to return the source address if the socket is unconnected. The caller should set *msg_namelen* to the size of this buffer before this call; upon return from a successful call, *msg_namelen* will contain the length of the returned address. If the application does not need to know the source address, *msg_name* can be specified as NULL.

The fields *msg_iov* and *msg_iovlen* describe scatter-gather locations, as discussed in readv(2).

The field *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*; upon return from a successful call it will contain the length of the control message sequence.

The messages are of the form:

```
struct cmsghdr {
size_t cmsg_len;    /* Data byte count, including header
(type is socklen_t in POSIX) */
int    cmsg_level;  /* Originating protocol */
int    cmsg_type;   /* Protocol-specific type */
/* followed by
unsigned char cmsg_data[]; */
};
```

Ancillary data should be accessed only by the macros defined in cmsg(3).

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

The *msg_flags* field in the *msghdr* is set on return of **recvmsg**(). It can contain several flags:

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 VALUE

These calls return the number of bytes received, or −1 if an error occurred. In the event of an error, *errno* is set to indicate the error.

When a stream socket peer has performed an orderly shutdown, the return value will be 0 (the traditional "end-of-file" return).

Datagram sockets in various domains (e.g., the UNIX and Internet domains) permit zero-length datagrams. When such a datagram is received, the return value is 0.

The value 0 may also be returned if the requested number of bytes to receive from a stream socket was 0.

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.

EAGAIN or **EWOULDBLOCK**
The socket is marked nonblocking and the receive operation would block, or a receive timeout had been set and the timeout expired before data was received. POSIX.1 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
The argument *sockfd* is an invalid file descriptor.

ECONNREFUSED
A remote host refused to allow the network connection (typically because it is not running the requested service).

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

EINTR
The receive was interrupted by delivery of a signal before any data were available; see signal(7).

EINVAL
Invalid argument passed.

ENOMEM
Could not allocate memory for **recvmsg**().

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

ENOTSOCK
The file descriptor *sockfd* does not refer to a socket.

CONFORMING TO

POSIX.1-2001, POSIX.1-2008, 4.4BSD (these interfaces first appeared in 4.2BSD).

POSIX.1 describes only the **MSG_OOB**, **MSG_PEEK**, and **MSG_WAITALL** flags.

NOTES

If a zero-length datagram is pending, read(2) and **recv**() with a *flags* argument of zero provide different behavior. In this circumstance, read(2) has no effect (the datagram remains pending), while **recv**() consumes the pending datagram.

The *socklen_t* type was invented by POSIX. See also accept(2).

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

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

EXAMPLE

An example of the use of **recvfrom**() is shown in getaddrinfo(3).

SEE ALSO

fcntl(2), getsockopt(2), read(2), recvmmsg(2), select(2), shutdown(2), socket(2), cmsg(3), sockatmark(3), ip(7), ipv6(7), socket(7), tcp(7), udp(7), unix(7)

COLOPHON

This page is part of release 4.16 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/.