NAME

udp − User Datagram Protocol for IPv4

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/udp.h>

udp_socket = socket(AF_INET, SOCK_DGRAM, 0);

DESCRIPTION

This is an implementation of the User Datagram Protocol described in RFC 768. It implements a connectionless, unreliable datagram packet service. Packets may be reordered or duplicated before they arrive. UDP generates and checks checksums to catch transmission errors.

When a UDP socket is created, its local and remote addresses are unspecified. Datagrams can be sent immediately using **sendto**(2) or **sendmsg**(2) with a valid destination address as an argument. When **connect**(2) is called on the socket, the default destination address is set and datagrams can now be sent using **send**(2) or **write**(2) without specifying a destination address. It is still possible to send to other destinations by passing an address to **sendto**(2) or **sendmsg**(2). In order to receive packets, the socket can be bound to a local address first by using **bind**(2). Otherwise, the socket layer will automatically assign a free local port out of the range defined by */proc/sys/net/ipv4/ip_local_port_range* and bind the socket to **INADDR_ANY**.

All receive operations return only one packet. When the packet is smaller than the passed buffer, only that much data is returned; when it is bigger, the packet is truncated and the **MSG_TRUNC** flag is set. **MSG_WAITALL** is not supported.

IP options may be sent or received using the socket options described in **ip**(7). They are processed by the kernel only when the appropriate */proc* parameter is enabled (but still passed to the user even when it is turned off). See **ip**(7).

When the **MSG_DONTROUTE** flag is set on sending, the destination address must refer to a local interface address and the packet is sent only to that interface.

By default, Linux UDP does path MTU (Maximum Transmission Unit) discovery. This means the kernel will keep track of the MTU to a specific target IP address and return **EMSGSIZE** when a UDP packet write exceeds it. When this happens, the application should decrease the packet size. Path MTU discovery can be also turned off using the **IP_MTU_DISCOVER** socket option or the */proc/sys/net/ipv4/ip_no_pmtu_disc* file; see **ip**(7) for details. When turned off, UDP will fragment outgoing UDP packets that exceed the interface MTU. However, disabling it is not recommended for performance and reliability reasons.

Address format

UDP uses the IPv4 *sockaddr_in* address format described in **ip**(7).

Error handling

All fatal errors will be passed to the user as an error return even when the socket is not connected. This includes asynchronous errors received from the network. You may get an error for an earlier packet that was sent on the same socket. This behavior differs from many other BSD socket implementations which don't pass any errors unless the socket is connected. Linux's behavior is mandated by **RFC 1122**.

For compatibility with legacy code, in Linux 2.0 and 2.2 it was possible to set the **SO_BSDCOMPAT SOL_SOCKET** option to receive remote errors only when the socket has been connected (except for **EPROTO** and **EMSGSIZE**). Locally generated errors are always passed. Support for this socket option was removed in later kernels; see **socket**(7) for further information.

When the **IP_RECVERR** option is enabled, all errors are stored in the socket error queue, and can be received by **recvmsg**(2) with the **MSG_ERRQUEUE** flag set.

/proc interfaces

System-wide UDP parameter settings can be accessed by files in the directory */proc/sys/net/ipv4/*.

udp_mem (since Linux 2.6.25)
This is a vector of three integers governing the number of pages allowed for queueing by all UDP sockets.

min Below this number of pages, UDP is not bothered about its memory appetite. When the amount of memory allocated by UDP exceeds this number, UDP starts to moderate memory usage.

pressure This value was introduced to follow the format of *tcp_mem* (see **tcp**(7)).

max Number of pages allowed for queueing by all UDP sockets.

Defaults values for these three items are calculated at boot time from the amount of available memory.

udp_rmem_min (integer; default value: PAGE_SIZE; since Linux 2.6.25)
Minimal size, in bytes, of receive buffers used by UDP sockets in moderation. Each UDP socket is able to use the size for receiving data, even if total pages of UDP sockets exceed *udp_mem* pressure.

udp_wmem_min (integer; default value: PAGE_SIZE; since Linux 2.6.25)
Minimal size, in bytes, of send buffer used by UDP sockets in moderation. Each UDP socket is able to use the size for sending data, even if total pages of UDP sockets exceed *udp_mem* pressure.

Socket options

To set or get a UDP socket option, call **getsockopt**(2) to read or **setsockopt**(2) to write the option with the option level argument set to **IPPROTO_UDP**. Unless otherwise noted, *optval* is a pointer to an *int*.

Following is a list of UDP-specific socket options. For details of some other socket options that are also applicable for UDP sockets, see **socket**(7).

UDP_CORK (since Linux 2.5.44)
If this option is enabled, then all data output on this socket is accumulated into a single datagram that is transmitted when the option is disabled. This option should not be used in code intended to be portable.

Ioctls

These ioctls can be accessed using **ioctl**(2). The correct syntax is:

int *value***;**
error **= ioctl(***udp_socket***,** *ioctl_type***, &***value***);**

FIONREAD (**SIOCINQ**)
Gets a pointer to an integer as argument. Returns the size of the next pending datagram in the integer in bytes, or 0 when no datagram is pending. **Warning:** Using **FIONREAD**, it is impossible to distinguish the case where no datagram is pending from the case where the next pending datagram contains zero bytes of data. It is safer to use **select**(2), **poll**(2), or **epoll**(7) to distinguish these cases.

TIOCOUTQ (**SIOCOUTQ**)
Returns the number of data bytes in the local send queue. Supported only with Linux 2.4 and above.

In addition, all ioctls documented in **ip**(7) and **socket**(7) are supported.

ERRORS

All errors documented for **socket**(7) or **ip**(7) may be returned by a send or receive on a UDP socket.

ECONNREFUSED
No receiver was associated with the destination address. This might be caused by a previous packet sent over the socket.

VERSIONS

IP_RECVERR is a new feature in Linux 2.2.

SEE ALSO

ip(7), **raw**(7), **socket**(7), **udplite**(7)

RFC 768 for the User Datagram Protocol.
RFC 1122 for the host requirements.
RFC 1191 for a description of path MTU discovery.

COLOPHON

This page is part of release 5.02 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/.