socket(2) System Calls Manual socket(2)

NAME

socket - create an endpoint for communication

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

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

int socket(int domain, int type, int protocol);
```

DESCRIPTION

socket() creates an endpoint for communication and returns a file de? scriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not cur? rently open for the process.

The domain argument specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. The formats currently understood by the Linux kernel include:

Name	Purpose	Man page

AF_LOCAL Synonym for AF_UNIX

AF_INET IPv4 Internet protocols ip(7)

AF_AX25 Amateur radio AX.25 protocol ax25(4)

AF_IPX IPX - Novell protocols

AF_APPLETALK AppleTalk ddp(7)

AF_X25 ITU-T X.25 / ISO/IEC 8208 protocol x25(7)

AF_INET6 IPv6 Internet protocols ipv6(7)

AF_DECnet DECet protocol sockets

AF_KEY Key management protocol, originally de?
veloped for usage with IPsec

AF_NETLINK Kernel user interface device netlink(7)

AF_PACKET Low-level packet interface packet(7)

AF_RDS Reliable Datagram Sockets (RDS) protocol rds(7)
rds-rdma(7)

AF_PPPOX Generic PPP transport layer, for setting
up L2 tunnels (L2TP and PPPoE)

AF_LLC Logical link control (IEEE 802.2 LLC)
protocol

AF_IB InfiniBand native addressing

AF_MPLS Multiprotocol Label Switching

AF_CAN Controller Area Network automotive bus
protocol

AF_TIPC TIPC, "cluster domain sockets" protocol

AF_BLUETOOTH Bluetooth low-level socket protocol

AF_VSOCK VSOCK (originally "VMWare VSockets") vsock(7)
protocol for hypervisor-guest communica?
tion
AF_KCM KCM (kernel connection multiplexer) in?
terface
AF_XDP XDP (express data path) interface

Further details of the above address families, as well as information on several other address families, can be found in address_families(7).

The socket has the indicated type, which specifies the communication semantics. Currently defined types are:

SOCK_STREAM Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mecha? nism may be supported.

SOCK_DGRAM Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK_SEQPACKET Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer is required to read an en? tire packet with each input system call.

SOCK_RAW Provides raw network protocol access.

SOCK_RDM Provides a reliable datagram layer that does not guar?
antee ordering.

SOCK_PACKET Obsolete and should not be used in new programs; see
packet(7).

Some socket types may not be implemented by all protocol families.

Since Linux 2.6.27, the type argument serves a second purpose: in addi?
tion to specifying a socket type, it may include the bitwise OR of any
of the following values, to modify the behavior of socket():

SOCK_NONBLOCK Set the O_NONBLOCK file status flag on the open file
description (see open(2)) referred to by the new file
descriptor. Using this flag saves extra calls to fc?
ntl(2) to achieve the same result.

SOCK_CLOEXEC Set the close-on-exec (FD_CLOEXEC) flag on the new file
descriptor. See the description of the O_CLOEXEC flag
in open(2) for reasons why this may be useful.

The protocol specifies a particular protocol to be used with the

socket type within a given protocol family, in which case protocol can be specified as 0. However, it is possible that many protocols may ex? ist, in which case a particular protocol must be specified in this man? ner. The protocol number to use is specific to the ?communication do? main? in which communication is to take place; see protocols(5). See getprotoent(3) on how to map protocol name strings to protocol numbers.

Sockets of type SOCK_STREAM are full-duplex byte streams. They do not preserve record boundaries. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a connect(2) call. Once connected, data may be transferred using read(2) and write(2) calls or some variant of the send(2) and recv(2) calls. When a session has been completed a close(2) may be performed. Out-of-band data may also be transmitted as described in send(2) and received as described in recv(2).

The communications protocols which implement a SOCK_STREAM ensure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered to be dead. When SO_KEEPALIVE is enabled on the socket the protocol checks in a protocol-specific manner if the other end is still alive. A SIG? PIPE signal is raised if a process sends or receives on a broken stream; this causes naive processes, which do not handle the signal, to

SOCK_STREAM sockets. The only difference is that read(2) calls will return only the amount of data requested, and any data remaining in the arriving packet will be discarded. Also all message boundaries in in? coming datagrams are preserved.

SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to corre? spondents named in sendto(2) calls. Datagrams are generally received with recvfrom(2), which returns the next datagram along with the ad? dress of its sender.

SOCK_PACKET is an obsolete socket type to receive raw packets directly from the device driver. Use packet(7) instead.

An fcntl(2) F_SETOWN operation can be used to specify a process or process group to receive a SIGURG signal when the out-of-band data ar? rives or SIGPIPE signal when a SOCK_STREAM connection breaks unexpect? edly. This operation may also be used to set the process or process group that receives the I/O and asynchronous notification of I/O events via SIGIO. Using F_SETOWN is equivalent to an ioctl(2) call with the FIOSETOWN or SIOCSPGRP argument.

When the network signals an error condition to the protocol module (e.g., using an ICMP message for IP) the pending error flag is set for the socket. The next operation on this socket will return the error

a per-socket error queue to retrieve detailed information about the er? ror; see IP_RECVERR in ip(7).

The operation of sockets is controlled by socket level options. These options are defined in <sys/socket.h>. The functions setsockopt(2) and getsockopt(2) are used to set and get options.

RETURN VALUE

On success, a file descriptor for the new socket is returned. On er? ror, -1 is returned, and errno is set to indicate the error.

ERRORS

EACCES Permission to create a socket of the specified type and/or pro? tocol is denied.

EAFNOSUPPORT

The implementation does not support the specified address fam? ily.

EINVAL Unknown protocol, or protocol family not available.

EINVAL Invalid flags in type.

EMFILE The per-process limit on the number of open file descriptors has

ENFILE The system-wide limit on the total number of open files has been reached.

ENOBUFS or ENOMEM
Insufficient memory is available. The socket cannot be created until sufficient resources are freed.

EPROTONOSUPPORT
The protocol type or the specified protocol is not supported within this domain.

Other errors may be generated by the underlying protocol modules.

STANDARDS

POSIX.1-2008.

SOCK_NONBLOCK and SOCK_CLOEXEC are Linux-specific.

HISTORY

POSIX.1-2001, 4.4BSD.

socket() appeared in 4.2BSD. It is generally portable to/from non-BSD systems supporting clones of the BSD socket layer (including System V

The manifest constants used under 4.x BSD for protocol families are PF_UNIX, PF_INET, and so on, while AF_UNIX, AF_INET, and so on are used for address families. However, already the BSD man page promises: "The protocol family generally is the same as the address family", and sub? sequent standards use AF_* everywhere.

EXAMPLES

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

SEE ALSO

accept(2), bind(2), close(2), connect(2), fcntl(2), getpeername(2), getsockname(2), getsockopt(2), ioctl(2), listen(2), read(2), recv(2), select(2), send(2), shutdown(2), socketpair(2), write(2), getpro? toent(3), address_families(7), ip(7), socket(7), tcp(7), udp(7), unix(7)

?An Introductory 4.3BSD Interprocess Communication Tutorial? and ?BSD Interprocess Communication Tutorial?, reprinted in UNIX Programmer's Supplementary Documents Volume 1.