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

The **socket**() system call creates an endpoint for communication and returns a descriptor.

The *domain* argument specifies a communications domain within which communication will take place; this selects the protocol family which should be used. These families are defined in the include file <*sys/socket.h*>. The currently understood formats are:

PF_LOCAL	Host-internal protocols (alias for PF_UNIX),
PF_UNIX	Host-internal protocols,
PF_INET	Internet version 4 protocols,
PF_INET6	Internet version 6 protocols,
PF_DIVERT	Firewall packet diversion/re-injection,
PF_ROUTE	Internal routing protocol,
PF_KEY	Internal key-management function,
PF_NETGRAPH	Netgraph sockets,
PF_NETLINK	Netlink protocols,
PF_BLUETOOTH	Bluetooth protocols,
PF_INET_SDP	OFED socket direct protocol (IPv4),
AF_HYPERV	HyperV sockets

Each protocol family is connected to an address family, which has the same name except that the prefix is "AF_" in place of "PF_". Other protocol families may be also defined, beginning with "PF_", with corresponding address families.

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

SOCK_STREAM	Stream socket,

SOCK_DGRAM	Datagram socket,
SOCK_RAW	Raw-protocol interface,
SOCK_SEQPACKET	Sequenced packet stream

A SOCK_STREAM type provides sequenced, reliable, two-way connection based byte streams. An out-of-band data transmission mechanism may be supported. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A SOCK_SEQPACKET socket may provide a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer may be required to read an entire packet with each read system call. This facility may have protocol-specific properties. SOCK_RAW sockets provide access to internal network protocols and interfaces. The SOCK_RAW type is available only to the super-user and is described in ip(4) and ip6(4).

Additionally, the following flags are allowed in the *type* argument:

SOCK_CLOEXEC	Set close-on-exec on the new descriptor,
SOCK_NONBLOCK	Set non-blocking mode on the new socket

The *protocol* argument specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place; see protocols(5).

The *protocol* argument may be set to zero (0) to request the default implementation of a socket type for the protocol, if any.

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. 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) system call. Once connected, data may be transferred using read(2) and write(2) calls or some variant of the send(2) and recv(2) functions. (Some protocol families, such as the Internet family, support the notion of an "implied connect", which permits data to be sent piggybacked onto a connect operation by using the sendto(2) system call.) 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 used to 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 broken and calls will indicate an error with -1 returns and with ETIMEDOUT as the specific code in the global variable *errno*.

The protocols optionally keep sockets "warm" by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for an extended period (e.g. 5 minutes). By default, a SIGPIPE signal is raised if a process sends on a broken stream, but this behavior may be inhibited via setsockopt(2).

SOCK_SEQPACKET sockets employ the same system calls as SOCK_STREAM sockets. The only difference is that read(2) calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.

SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in send(2) calls. Datagrams are generally received with recvfrom(2), which returns the next datagram with its return address.

An fcntl(2) system call can be used to specify a process group to receive a SIGURG signal when the out-of-band data arrives. It may also enable non-blocking I/O and asynchronous notification of I/O events via SIGIO.

The operation of sockets is controlled by socket level *options*. These options are defined in the file <*sys/socket.h*>. The setsockopt(2) and getsockopt(2) system calls are used to set and get options, respectively.

RETURN VALUES

A -1 is returned if an error occurs, otherwise the return value is a descriptor referencing the socket.

ERRORS

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

[EACCES]	Permission to create a socket of the specified type and/or protocol is denied.
[EAFNOSUPPORT]	The address family (domain) is not supported or the specified domain is not supported by this protocol family.
[EMFILE]	The per-process descriptor table is full.
[ENFILE]	The system file table is full.
[ENOBUFS]	Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.
[EPERM]	User has insufficient privileges to carry out the requested operation.

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

[EPROTOTYPE] The socket type is not supported by the protocol.

SEE ALSO

accept(2), bind(2), connect(2), divert(4), getpeername(2), getsockname(2), getsockopt(2), ioctl(2), ip(4), ip6(4), listen(2), read(2), recv(2), select(2), send(2), shutdown(2), socketpair(2), write(2), CMSG_DATA(3), getprotoent(3), netgraph(4), protocols(5)

"An Introductory 4.3 BSD Interprocess Communication Tutorial", *PS1*, 7.

"BSD Interprocess Communication Tutorial", *PS1*, 8.

STANDARDS

The **socket**() function conforms to IEEE Std 1003.1-2008 ("POSIX.1"). The POSIX standard specifies only the AF_INET, AF_INET6, and AF_UNIX constants for address families, and requires the use of AF_* constants for the *domain* argument of **socket**(). The SOCK_CLOEXEC flag is expected to conform to the next revision of the POSIX standard. The SOCK_RDM *type*, the PF_* constants, and other address families are FreeBSD extensions.

HISTORY

The **socket**() system call appeared in 4.2BSD.