NAME

socket – create an endpoint for communication

SYNOPSIS

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

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

DESCRIPTION

Socket creates an endpoint for communication and returns a descriptor.

The *domain* parameter 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

AF_UNIX	(UNIX internal protocols)
AF_INET	(ARPA Internet protocols)
AF_ISO	(ISO protocols)
AF_NS	(Xerox Network Systems protocols)
AF_IMPLINK	(IMP "host at IMP" link layer)

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

SOCK_STREAM
SOCK_DGRAM
SOCK_RAW
SOCK_SEQPACKET
SOCK_RDM

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 is protocol specific, and presently implemented only for **AF_NS**. **SOCK_RAW** sockets provide access to internal network protocols and interfaces. The types **SOCK_RAW**, which is available only to the super-user, and **SOCK_RDM**, which is planned, but not yet implemented, are not described here.

The *protocol* 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).

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) 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 used to implement a **SOCK_STREAM** insure 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 a extended period (e.g. 5 minutes). A **SIGPIPE** signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

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) 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*. **Setsockopt**(2) and **getsockopt**(2) 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

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

EMFILE The per-process descriptor table is full.

ENFILE The system file table is full.

EACCESS Permission to create a socket of the specified type and/or protocol is denied.

ENOBUFS Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.

CONFORMING TO

4.4BSD (the **socket** function call appeared in 4.2BSD). Generally portable to/from non-BSD systems supporting clones of the BSD socket layer (including System V variants).

SEE ALSO

accept(2), **bind**(2), **connect**(2), **getprotoent**(3), **getsockname**(2), **getsockopt**(2), **ioctl**(2), **listen**(2), **read**(2), **recv**(2), **select**(2), **send**(2), **shutdown**(2), **socketpair**(2), **write**(2)

"An Introductory 4.3 BSD Interprocess Communication Tutorial" is reprinted in *UNIX Programmer's Supplementary Documents Volume 1*

"BSD Interprocess Communication Tutorial" is reprinted in *UNIX Programmer's Supplementary Documents Volume 1*