NAME

tcp, TCP – Internet Transmission Control Protocol

SYNOPSIS

#include <sys/socket.h>

#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);

t = t_open("/dev/tcp", O_RDWR);

DESCRIPTION

TCP is the virtual circuit protocol of the Internet protocol family. It provides reliable, flow-controlled, in order, two-way transmission of data. It is a byte-stream protocol layered above the Internet Protocol (IP), the Internet protocol family's internetwork datagram delivery protocol.

Programs can access TCP using the socket interface as a **SOCK_STREAM** socket type, or using the Transport Level Interface (TLI) where it supports the connection-oriented (**T_COTS_ORD**) service type.

TCP uses IP's host-level addressing and adds its own per-host collection of "port addresses." The endpoints of a TCP connection are identified by the combination of an IP address and a TCP port number. Although other protocols, such as the User Datagram Protocol (UDP), may use the same host and port address format, the port space of these protocols is distinct. See **inet**(7P) for details on the common aspects of addressing in the Internet protocol family.

Sockets utilizing TCP are either "active" or "passive". Active sockets initiate connections to passive sockets. Both types of sockets must have their local IP address and TCP port number bound with the **bind**(3N) system call after the socket is created. By default, TCP sockets are active. A passive socket is created by calling the **listen**(3N) system call after binding the socket with **bind( )**. This establishes a queueing parameter for the passive socket. After this, connections to the passive socket can be received with the **accept**(3N) system call. Active sockets use the **connect**(3N) call after binding to initiate connections.

By using the special value **INADDR_ANY**, the local IP address can be left unspecified in the **bind( )** call by either active or passive TCP sockets. This feature is usually used if the local address is either unknown or irrelevant. If left unspecified, the local IP address will be bound at connection time to the address of the network interface used to service the connection.

Once a connection has been established, data can be exchanged using the **read**(2) and **write**(2) system calls.

TCP supports one socket option which is set with **setsockopt( )** and tested with **getsockopt**(3N). Under most circumstances, TCP sends data when it is presented. When outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays.

Therefore, TCP provides a boolean option, **TCP_NODELAY** (defined in **<netinet/tcp.h>**), to defeat this algorithm. The option level for the **setsockopt( )** call is the protocol number for TCP, available from **getprotobyname**(3N).

Options at the IP level may be used with TCP. See **ip**(7P).

TCP provides an urgent data mechanism, which may be invoked using the out-of-band provisions of **send**(3N). The caller may mark one byte as "urgent" with the **MSG_OOB** flag to **send**(3N). This sets an "urgent pointer" pointing to this byte in the TCP stream. The receiver on the other side of the stream is notified of the urgent data by a **SIGURG** signal. The **SIOCATMARK ioctl( )** request returns a value indicating whether the stream is at the urgent mark. Because the system never returns data across the urgent mark in a single **read**(2) call, it is possible to advance to the urgent data in a simple loop which reads data, testing the socket with the **SIOCATMARK ioctl( )** request, until it reaches the mark.

Incoming connection requests that include an IP source route option are noted, and the reverse source route is used in responding.

A checksum over all data helps TCP implement reliability. Using a window-based flow control mechanism that makes use of positive acknowledgements, sequence numbers, and a retransmission strategy, TCP can usually recover when datagrams are damaged, delayed, duplicated or delivered out of order by the underlying communication medium.

If the local TCP receives no acknowledgements from its peer for a period of time, as would be the case if the remote machine crashed, the connection is closed and an error is returned to the user. If the remote machine reboots or otherwise loses state information about a TCP connection, the connection is aborted and an error is returned to the user.

SEE ALSO

read(2), **write**(2), **accept**(3N), **bind**(3N), **connect**(3N), **getprotobyname**(3N), **getsockopt**(3N), **listen**(3N), **send**(3N), **inet**(7P), **ip**(7P)

Postel, Jon, *Transmission Control Protocol - DARPA Internet Program Protocol Specification*, RFC 793, Network Information Center, SRI International, Menlo Park, Calif., September 1981.

DIAGNOSTICS

A socket operation may fail if:

EISCONN A **connect( )** operation was attempted on a socket on which a **connect( )** operation had already been performed.

ETIMEDOUT A connection was dropped due to excessive retransmissions.

ECONNRESET The remote peer forced the connection to be closed (usually because the remote machine has lost state information about the connection due to a crash).

ECONNREFUSED The remote peer actively refused connection establishment (usually because no process is listening to the port).

EADDRINUSE A **bind( )** operation was attempted on a socket with a network address/port pair that has already been bound to another socket.

EADDRNOTAVAIL A **bind( )** operation was attempted on a socket with a network address for which no network interface exists.

EACCES A **bind( )** operation was attempted with a “reserved” port number and the effective user ID of the process was not the privileged user.

ENOBUFS The system ran out of memory for internal data structures.