# The `sock` Program

A simple test program named sock is used throughout the book to generate TCP and UDP data. It is used as both a client and server process. Having a test program like this, which is executable from a shell prompt, prevents us from having to write new client and server C programs for each specific feature that we want to examine. Since the purpose of this book is to understand the networking protocols, and not network programming, in this Appendix we only describe the program and its various options.

There are numerous other programs with functionality similar to sock. Juergen Nickelsen wrote a program named socket and Dave Yost wrote a program named sockio. Both contain many similar features. Pieces of the *sock* program have also been inspired by the public domain ttcp program, written by Mike Muuss and Terry Slattery.

The sock program operates in one of four modes:

1. Interactive client: the default. The program connects to a server and then copies standard input to the server and copies everything received from the server to standard output. This is shown in Figure C.1.

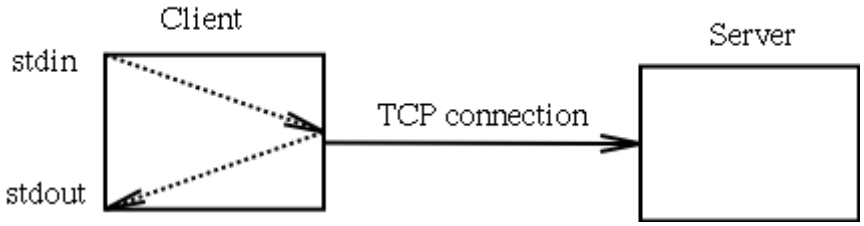


**Figure C.1** Default operation of sock as interactive client.

We must specify the name of the server host and the name of the service to connect to. The host can also be specified as a dotted-decimal number, and the service can be specified as an integer port number. Connecting to the standard echo server (Section 1.12), from sun to bsdi echoes everything we type:

```
sun % sock bsdi echo
a test line                 we type this line
a test line                 and the echo server returns a copy
^D                          type our end-of-file character to terminate
```

2. Interactive server: the -s option is specified. The service name (or port number) is required:
   `sun % sock -s 5555`  *act as server listening on port 5555*
3. The program waits for a connection from a client and then copies standard input to the client and copies everything received from the client to standard output. An Internet address can precede the port number on the command line, to specify on which local interface connections are accepted:
   sun % **sock -s 140.252.13.33 5555**  *accept connections only on Ethernet*
4. The default mode is to accept a connection request on any local interface.
5. Source client: the -i option is specified. By default a 1024-byte buffer is written to the network 1024 times. The -n and -w options can change these defaults. For example,

```
sun % sock -i -nl2 -w4096 bsdi discard
```

writes 12 buffers, each containing 4096 bytes of data, to the discard server on host bsdi.

6. Sink server: the -i and -s options are specified. Data is read from the network and discarded.

Although these examples used TCP (the default), the -u option specifies UDP.

There are a multitude of options that provide finer control over exactly how the program operates. These options are needed to generate all the test conditions used throughout the text.

| | |
|---|---|
| -b *n* | Bind *n* as the client's local port number. (By default an ephemeral port number assigned by the system is used by the client.) |
| -c | Convert newline characters that are read on standard input into a carriage return and a linefeed. Similarly, when reading from the network, convert the sequence <carriage return, linefeed> into a single newline character. Many Internet applications expect NVT ASCII (Section 26.4), which uses the carriage return and line-feed to terminate each line. |
| -f *a.b.c.d.p* | Specify the foreign IP address *(a.b.c.d)* and the foreign port number *(p)* for a UDP end point (Section 11.12). |
| -h | Implement TCP's half-close facility (Section 18.5). That is, do not terminate when an end-of-file is encountered on standard input. Instead, issue a half-close on the TCP connection but continue reading from the network until the peer closes the connection. |
| -i | Source client or sink server. Either write data to the network (default) or if used in conjunction with the -s option, read data from the network. The -n option can specify the number of buffers to write (or read), the -w option can specify the size of each write, and the -r option can specify the size of each read. |
| -n *n* | When used with the -i option, *n* specifies the number of buffers to read or write. The default value of *n* is 1024. |
| -p *n* | Specify the number of seconds to pause between each read or write. This can be used with the source client (-i) or sink server (-is) to delay between each read or write of network. Also see the -P option to pause before the first read or write. |
| -q *n* | Specify the size of the pending connection queue for the TCP server: the number of accepted connections that TCP will queue for the application (Figure 18.23). The default is 5. |
| -r *n* | When used with the -is options, *n* specifies the size of each read from the network. The default is 1024 bytes per read. |
| -s | Operate as a server instead of as a client. |
| -u | Use UDP instead of TCP. |
| -v | Verbose. Print additional details (such as the client and server ephemeral port numbers) onto standard error. |
| -w *n* | When used with the -i option, specifies the size of each write to the network. The default is 1024 bytes per write. |
| -A | Enable the SO_REUSEADDR socket option. With TCP this allows the process to assign itself a port number that is part of a connection that is in the 2MSL wait. With UDP on a system that supports multicasting, it allows multiple processes to use the same local port to receive broadcast or multicast datagrams. |
| -B | Enable the SO_BROADCAST socket option to allow UDP datagrams to be sent to a broadcast IP address. |
| -D | Enable the SO_DEBUG socket option. This causes additional debugging information to be maintained by the kernel for this TCP connection (Section A.6). This information can be output later by running the trpt(8) program. |

| | |
|---|---|
| -E | Enable the IP_RECVDSTADDR socket option, if supported by the implementation (Section 11.12). This is intended for UDP servers, to print the destination IP address of the received UDP datagram. |
| -F | Specifies a concurrent TCP server. That is, the server creates a new process using the fork function for each client connection. |
| -K | Enable TCP's SO_KEEPALIVE socket option (Chapter 23). |
| -L *n* | Set the linger time (SO_LINGER socket option) for a TCP end point to *n.* A linger time of 0 means when the network connection is closed, any data still queued for sending is discarded and a reset is sent to the peer (Section 18.7). A positive linger time is the time (in 100ths of a second) that a close on the network connection should wait for all outstanding data to be sent and acknowledged. If, after closing the network connection, all the pending data has not been sent and acknowledged when this timer expires, the close will return an error. |
| -N | Set the TCP_NODELAY socket option to disable the Nagle algorithm (Section 19.4). |
| -O *n* | Specify the number of seconds for a TCP server to pause before accepting the first client connection. |
| -P *n* | Specify the number of seconds to pause before the first read or write of the network. This can be used with the sink server (-is) to delay after accepting the connection request from the client but before performing the first read from the network. When used with the source client (-i) it delays after the connection has been established, but before the first write to the network. Also see the -p option to pause between each successive read or write. |
| -Q *n* | Specify the number of seconds for a TCP client or server to pause after receiving an end-of-file from the other end, but before closing its end of the connection. |
| -R *n* | Set the socket's receive buffer (SO_RCVBUF socket option) to *n.* This can directly affect the size of the receive window advertised by TCP. With UDP this specifies the largest UDP datagram that can be received. |
| -S *n* | Set the socket's send buffer (SO_SNDBUF socket option) to *n.* With UDP this specifies the largest UDP datagram that can be sent. |
| -U *n* | Enter TCP's urgent mode after write number *n* to the network. One byte of data is written to initiate urgent mode (Section 20.8). |

**Documentación extraída del libro “TCP/IP Illustrated”, Volume 1: The protocols, W Richard Stevens, Addison-Wesley, 1994**

**http://www.uic.rsu.ru/doc/inet/tcp_stevens/append_c.htm**