## 4.3 connect Function

The connect function is used by a TCP client to establish a connection with a TCP server.

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

int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

Returns: 0 if OK, -1 on error

*sockfd* is a socket descriptor returned by the socket function. The second and third arguments are a pointer to a socket address structure and its size, as described in Section 3.3. The socket address structure must contain the IP address and port number of the server. We saw an example of this function in Figure 1.5.

The client does not have to call bind (which we will describe in the next section) before calling connect: the kernel will choose both an ephemeral port and the source IP address if necessary.

In the case of a TCP socket, the connect function initiates TCP's three-way handshake (Section 2.6). The function returns only when the connection is established or an error occurs. There are several different error returns possible.

1. If the client TCP receives no response to its SYN segment, ETIMEDOUT is returned. 4.4BSD, for example, sends one SYN when connect is called, another 6 seconds later, and another 24 seconds later (p. 828 of TCPv2). If no response is received after a total of 75 seconds, the error is returned.

   Some systems provide administrative control over this timeout; see Appendix E of TCPv1.

2. If the server's response to the client's SYN is a reset (RST), this indicates that no process is waiting for connections on the server host at the port specified (i.e., the server process is probably not running). This is a *hard error* and the error ECONNREFUSED is returned to the client as soon as the RST is received.

   An RST is a type of TCP segment that is sent by TCP when something is wrong. Three conditions that generate an RST are: when a SYN arrives for a port that has no listening server (what we just described), when TCP wants to abort an existing connection, and when TCP receives a segment for a connection that does not exist. (TCPv1 [pp. 246–250] contains additional information.)

3. If the client's SYN elicits an ICMP "destination unreachable" from some intermediate router, this is considered a *soft error*. The client kernel saves the message but keeps sending SYNs with the same time between each SYN as in the first scenario. If no response is received after some fixed amount of time (75 seconds for 4.4BSD), the saved ICMP error is returned to the process as either EHOSTUNREACH or ENETUNREACH. It is also possible that the remote system is not reachable by any route in the local system's forwarding table, or that the connect call returns without waiting at all.

   > Many earlier systems, such as 4.2BSD, incorrectly aborted the connection establishment attempt when the ICMP "destination unreachable" was received. This is wrong because this ICMP error can indicate a transient condition. For example, it could be that the condition is caused by a routing problem that will be corrected.
   >
   > Notice that ENETUNREACH is not listed in Figure A.15, even when the error indicates that the destination network is unreachable. Network unreachables are considered obsolete, and applications should just treat ENETUNREACH and EHOSTUNREACH as the same error.

We can see these different error conditions with our simple client from Figure 1.5. We first specify the local host (127.0.0.1), which is running the daytime server, and see the output.

```
solaris % daytimetcpcli 127.0.0.1
Sun Jul 27 22:01:51 2003
```

To see a different format for the returned reply, we specify a different machine's IP address (in this example, the IP address of the HP-UX machine).

```
solaris % daytimetcpcli 192.6.38.100
Sun Jul 27 22:04:59 PDT 2003
```

Next, we specify an IP address that is on the local subnet (192.168.1/24) but the host ID (100) is nonexistent. That is, there is no host on the subnet with a host ID of 100, so when the client host sends out ARP requests (asking for that host to respond with its hardware address), it will never receive an ARP reply.

```
solaris % daytimetcpcli 192.168.1.100
connect error: Connection timed out
```

We only get the error after the connect times out (around four minutes with Solaris 9). Notice that our err_sys function prints the human-

readable string associated with the ETIMEDOUT error.

Our next example is to specify a host (a local router) that is not running a daytime server.

```
solaris % daytimetcpcli 192.168.1.5
connect error: Connection refused
```

The server responds immediately with an RST.

Our final example specifies an IP address that is not reachable on the Internet. If we watch the packets with tcpdump, we see that a router six hops away returns an ICMP host unreachable error.

```
solaris % daytimetcpcli 192.3.4.5
connect error: No route to host
```

As with the ETIMEDOUT error, in this example, connect returns the EHOSTUNREACH error only after waiting its specified amount of time.

In terms of the TCP state transition diagram (Figure 2.4), connect moves from the CLOSED state (the state in which a socket begins when it is created by the socket function) to the SYN_SENT state, and then, on success, to the ESTABLISHED state. If connect fails, the socket is no longer usable and must be closed. We cannot call connect again on the socket. In Figure 11.10, we will see that when we call connect in a loop, trying each IP address for a given host until one works, each time connect fails, we must close the socket descriptor and call socket again.



PREVIOUS NEXT