4.4 bind Function

The bind function assigns a local protocol address to a socket. With the Internet protocols, the protocol address is the combination of either a 32-bit IPv4 address or a 128-bit IPv6 address, along with a 16-bit TCP or UDP port number.

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

int bind (int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);
```

Returns: 0 if OK,-1 on error

> Historically, the man page description of bind has said "bind assigns a name to an unnamed socket." The use of the term "name" is confusing and gives the connotation of domain names (Chapter 11) such as foo.bar.com. The bind function has nothing to do with names. bind assigns a protocol address to a socket, and what that protocol address means depends on the protocol.

The second argument is a pointer to a protocol-specific address, and the third argument is the size of this address structure. With TCP, calling bind lets us specify a port number, an IP address, both, or neither.

- Servers bind their well-known port when they start. We saw this in Figure 1.9. If a TCP client or server does not do this, the kernel chooses an ephemeral port for the socket when either connect or listen is called. It is normal for a TCP client to let the kernel choose an ephemeral port, unless the application requires a reserved port (Figure 2.10), but it is rare for a TCP server to let the kernel choose an ephemeral port, since servers are known by their well-known port.

  > Exceptions to this rule are Remote Procedure Call (RPC) servers. They normally let the kernel choose an ephemeral port for their listening socket since this port is then registered with the RPC port mapper. Clients have to contact the port mapper to obtain the ephemeral port before they can connect to the server. This also applies to RPC servers using UDP.

- A process can bind a specific IP address to its socket. The IP address must belong to an interface on the host. For a TCP client, this assigns the source IP address that will be used for IP datagrams sent on the socket. For a TCP server, this restricts the socket to receive incoming client connections destined only to that IP address.

  Normally, a TCP client does not bind an IP address to its socket. The kernel chooses the source IP address when the socket is connected, based on the outgoing interface that is used, which in turn is based on the route required to reach the server (p. 737 of TCPv2).

  If a TCP server does not bind an IP address to its socket, the kernel uses the destination IP address of the client's SYN as the server's source IP address (p. 943 of TCPv2).

As we said, calling bind lets us specify the IP address, the port, both, or neither. Figure 4.6 summarizes the values to which we set sin_addr and sin_port, or sin6_addr and sin6_port, depending on the desired result.

Figure 4.6. Result when specifying IP address and/or port number to bind.

Process specifies		Result
IP address	port	
Wildcard	0	Kernel chooses IP address and port
Wildcard	nonzero	Kernel chooses IP address, process specifies port
Local IP address	0	Process specifies IP address, kernel chooses port
Local IP address	nonzero	Process specifies IP address and port

If we specify a port number of 0, the kernel chooses an ephemeral port when bind is called. But if we specify a wildcard IP address, the kernel does not choose the local IP address until either the socket is connected (TCP) or a datagram is sent on the socket (UDP).

With IPv4, the *wildcard* address is specified by the constant INADDR_ANY, whose value is normally 0. This tells the kernel to choose the IP address. We saw the use of this in Figure 1.9 with the assignment

```
struct sockaddr_in   servaddr;
servaddr.sin_addr.s_addr = htonl (INADDR_ANY);     /* wildcard */
```

While this works with IPv4, where an IP address is a 32-bit value that can be represented as a simple numeric constant (0 in this case), we cannot use this technique with IPv6, since the 128-bit IPv6 address is stored in a structure. (In C we cannot represent a constant structure

on the right-hand side of an assignment.) To solve this problem, we write

```
struct sockaddr_in6    serv;

serv.sin6_addr = in6addr_any;     /* wildcard */
```

The system allocates and initializes the `in6addr_any` variable to the constant `IN6ADDR_ANY_INIT`. The `<netinet/in.h>` header contains the `extern` declaration for `in6addr_any`.

The value of `INADDR_ANY` (0) is the same in either network or host byte order, so the use of `htonl` is not really required. But, since all the `INADDR_`constants defined by the `<netinet/in.h>` header are defined in host byte order, we should use `htonl` with any of these constants.

If we tell the kernel to choose an ephemeral port number for our socket, notice that `bind` does not return the chosen value. Indeed, it cannot return this value since the second argument to `bind` has the `const` qualifier. To obtain the value of the ephemeral port assigned by the kernel, we must call `getsockname` to return the protocol address.

A common example of a process binding a non-wildcard IP address to a socket is a host that provides Web servers to multiple organizations (Section 14.2 of TCPv3). First, each organization has its own domain name, such as `www.`*organization*`.com`. Next, each organization's domain name maps into a different IP address, but typically on the same subnet. For example, if the subnet is 198.69.10, the first organization's IP address could be 198.69.10.128, the next 198.69.10.129, and so on. All these IP addresses are then *aliased* onto a single network interface (using the `alias` option of the `ifconfig` command on 4.4BSD, for example) so that the IP layer will accept incoming datagrams destined for any of the aliased addresses. Finally, one copy of the HTTP server is started for each organization and each copy `binds` only the IP address for that organization.

> An alternative technique is to run a single server that binds the wildcard address. When a connection arrives, the server calls `getsockname` to obtain the destination IP address from the client, which in our discussion above could be 198.69.10.128, 198.69.10.129, and so on. The server then handles the client request based on the IP address to which the connection was issued.
>
> One advantage in binding a non-wildcard IP address is that the demultiplexing of a given destination IP address to a given server process is then done by the kernel.
>
> We must be careful to distinguish between the interface on which a packet arrives versus the destination IP address of that packet. In Section 8.8, we will talk about the weak end system model and the strong end system model. Most implementations employ the former, meaning it is okay for a packet to arrive with a destination IP address that identifies an interface other than the interface on which the packet arrives. (This assumes a multihomed host.) Binding a non-wildcard IP address restricts the datagrams that will be delivered to the socket based only on the destination IP address. It says nothing about the arriving interface, unless the host employs the strong end system model.

A common error from `bind` is `EADDRINUSE` ("Address already in use"). We will say more about this in Section 7.5 when we talk about the `SO_REUSEADDR` and `SO_REUSEPORT` socket options.