TCP port numbers and concurrent servers

With a concurrent server, where the main server loop spawns a child to handle each new connection, what happens if the child continues to use the well-known port number while serving a long request? Let’s examine a typical sequence. First, the server is started on the host *bsdi,* which is multihomed with IP addresses 206.62.226.35 and 206.62.226.66 and the server does a passive open using its well-known port number(21, for this example). It is now waiting for a client request which we show in figure:



We use the notation {\*.21, \*.\*} to indicate the server’s socket pair. The server is waiting for a connection request on any local interface(the first asterisk), on port 21. The foreign IP address and foreign port are not specified and we denote them as \*.\*. We also call this a *listening socket.*

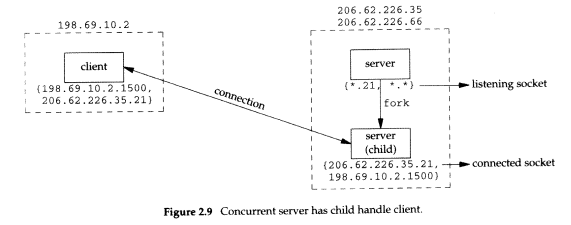
We use a period to separate the IP address from the port number that is what *netstat* use. This is sometimes confusing because decimal points are used both domain name(solaris.kohala.com.21) and in Ipv4 dotted-decimal notation(206.62.226.33.21).

When we specify the local IP address as an asterisk, it is called the *wildcard* character. If the host on which the server is running is multihomed(as in this example), the server can specify that it want only to accept incoming connections that arrive destined to one specified local interface. This is a one-or-any choice for the server. The server cannot specify a list of multiple addresses. The wildcard addresses was specified by setting the IP address in the socket address structure to INADDR\_ANY before calling ***bind***.

At some latter time a client starts on the host with IP address 198.69.10.2 and executes an active open to the server’s IP address of 206.62.226.35. We assume the empheral port chosen by the client TCP is 1500 for this example(figure 2.8). Beneath the client we show its socket pair.



When the server receives and accepts the client’s connection, it *fork*s a copy of itself, letting the child handle the client, as we shown in figure 2.9(we describe the fork function in section 4.7)



At this point we must distinguish on the server host between the listening socket and the connected socket. Notice that the connected sockt uses the same local port(21) as used for the listening socket. Also notice that on the multihomed server the local address is filled in for the connected socket(206.62.226.35) once the connection is established.

The next step assumes that another client process on the client host requests a connection with the same server. The TCP code in the client host assigns the new client socket an unused enphemeral port number, say 1501. This gives us the scenario shown in figure 2.10. On the server the two connections are distinct: the socket pair for the first connection differs from the socket pair for the second connection because the client’s TCP chooses an unused port for the second connection(1501).

Notice from this example that TCP cannot demultiplex incoming segments by looking at just the destination port number. TCP must look at full for elements in the socket pair to determine which endpoint receives an arriving segment. In figure 2.10 we have three sockets with the same local port(21). If a segment arrives from 198.69.10.2 port 1500 destined for 206.62.266.35 port 21, it is delivered to the first child. If a segment arrives from 198.69.10.2 port 1501 destined for 206.62.266.35 port 21, it is delivered to the second child. All other TCP segments destined for port 21 are delivered to the original server with the listening socket.

