5.2 TCP Echo Server: `main` Function

Our TCP client and server follow the flow of functions that we diagrammed in Figure 4.1. We show the concurrent server program in Figure 5.2.

Create socket, bind server's well-known port

9–15 A TCP socket is created. An Internet socket address structure is filled in with the wildcard address (`INADDR_ANY`) and the server's well-known port (`SERV_PORT`, which is defined as 9877 in our `unp.h` header). Binding the wildcard address tells the system that we will accept a connection destined for any local interface, in case the system is multihomed. Our choice of the TCP port number is based on Figure 2.10. It should be greater than 1023 (we do not need a reserved port), greater than 5000 (to avoid conflict with the ephemeral ports allocated by many Berkeley-derived implementations), less than 49152 (to avoid conflict with the "correct" range of ephemeral ports), and it should not conflict with any registered port. The socket is converted into a listening socket by `listen`.

Wait for client connection to complete

17–18 The server blocks in the call to `accept`, waiting for a client connection to complete.

Concurrent server

19–24 For each client, `fork` spawns a child, and the child handles the new client. As we discussed in Section 4.8, the child closes the listening socket and the parent closes the connected socket. The child then calls `str_echo` (Figure 5.3) to handle the client.

Figure 5.2 TCP echo server (improved in Figure 5.12).

tcpdiserv/tcpserv01.c

```
 1 #include      "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     listenfd, connfd;
 6     pid_t   childpid;
 7     socklen_t clilen;
 8     struct sockaddr_in cliaddr, servaddr;

 9     listenfd = Socket (AF_INET, SOCK_STREAM, 0);

10     bzero(&servaddr, sizeof(servaddr));
11     servaddr.sin_family = AF_INET;
12     servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
13     servaddr.sin_port = htons (SERV_PORT);

14     Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));

15     Listen(listenfd, LISTENQ);

16     for ( ; ; )  {
17         clilen = sizeof(cliaddr);
18         connfd = Accept(listenfd, (SA *) &cliaddr, &clilen);

19         if ( (childpid = Fork()) == 0) { /* child process */
20             Close(listenfd);    /* close listening socket */
21             str_echo(connfd);   /* process the request */
22             exit (0);
23         }
24         Close(connfd);          /* parent closes connected socket */
25     }
26 }
```