8.15 TCP and UDP Echo Server Using `select`

We now combine our concurrent TCP echo server from Chapter 5 with our iterative UDP echo server from this chapter into a single server that uses `select` to multiplex a TCP and UDP socket. Figure 8.24 is the first half of this server.

Create listening TCP socket

14–22 A listening TCP socket is created that is bound to the server's well-known port. We set the `SO_REUSEADDR` socket option in case connections exist on this port.

Create UDP socket

23–29 A UDP socket is also created and bound to the same port. Even though the same port is used for TCP and UDP sockets, there is no need to set the `SO_REUSEADDR` socket option before this call to `bind`, because TCP ports are independent of UDP ports.

Figure 8.25 shows the second half of our server.

Establish signal handler for `SIGCHLD`

30 A signal handler is established for `SIGCHLD` because TCP connections will be handled by a child process. We showed this signal handler in Figure 5.11.

Prepare for `select`

31–32 We initialize a descriptor set for `select` and calculate the maximum of the two descriptors for which we will wait.

Figure 8.24 First half of echo server that handles TCP and UDP using `select`.

udpcliserv/udpservselect01.c

```
 1 #include     "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     listenfd, connfd, udpfd, nready, maxfdp1;
 6     char    mesg[MAXLINE];
 7     pid_t   childpid;
 8     fd_set  rset;
 9     ssize_t n;
10     socklen_t len;
11     const int on = 1;
12     struct sockaddr_in cliaddr, servaddr;
13     void    sig_chld(int);

14         /* create listening TCP socket */
15     listenfd = Socket(AF_INET, SOCK_STREAM, 0);

16     bzero(&servaddr, sizeof(servaddr));
17     servaddr.sin_family = AF_INET;
18     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
19     servaddr.sin_port = htons(SERV_PORT);

20     Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
21     Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));

22     Listen(listenfd, LISTENQ);

23          /* create UDP socket */
24     udpfd = Socket(AF_INET, SOCK_DGRAM, 0);

25     bzero(&servaddr, sizeof(servaddr));
26     servaddr.sin_family = AF_INET;
27     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
28      servaddr.sin_port = htons(SERV_PORT);

29      Bind(udpfd, (SA *) &servaddr, sizeof(servaddr));
```

Call `select`

34–41 We call `select`, waiting only for readability on the listening TCP socket or readability on the UDP socket. Since our `sig_chld` handler can interrupt our call to `select`, we handle an error of `EINTR`.

Handle new client connection

42–51 We `accept` a new client connection when the listening TCP socket is readable, `fork` a child, and call our `str_echo` function in the child. This is the same sequence of steps we used in Chapter 5.

Figure 8.25 Second half of echo server that handles TCP and UDP using `select`.

udpcliserv/udpservselect01.c

```
30      Signal(SIGCHLD, sig_chld);      /* must call waitpid() */

31      FD_ZERO(&rset);
32      maxfdp1 = max(listenfd, udpfd) + 1;
33      for ( ; ; ) {
34          FD_SET(listenfd, &rset);
35          FD_SET(udpfd, &rset);
36          if ( (nready = select(maxfdp1, &rset, NULL, NULL, NULL)) < 0) {
37              if (errno == EINTR)
38                  continue;      /* back to for() */
39              else
40                  err_sys("select error");
41           }

42           if (FD_ISSET(listenfd, &rset)) {
43               len = sizeof(cliaddr);
44               connfd = Accept(listenfd, (SA *) &cliaddr, &len);

45               if ( (childpid = Fork()) == 0) { /* child process */
46                   Close(listenfd);      /* close listening socket */
47                   str_echo(connfd);     /* process the request */
48                   exit(0);
49                }
50                Close(connfd);      /* parent closes connected socket */
51          }

52          if (FD_ISSET(udpfd, &rset)) {
53              len = sizeof(cliaddr);
54              n = Recvfrom(udpfd, mesg, MAXLINE, 0, (SA *) &cliaddr, &len);

55              Sendto(udpfd, mesg, n, 0, (SA *) &cliaddr, len);
56          }
57      }
58  }
```

Handle arrival of datagram

52–57 If the UDP socket is readable, a datagram has arrived. We read it with `recvfrom` and send it back to the client with `sendto`.