10.2 SCTP One-to-Many-Style Streaming Echo Server: `main` Function

Our SCTP client and server follow the flow of functions diagrammed in Figure 9.2. We show an iterative server program in Figure 10.2.

Set stream increment option

13–14 By default, our server responds using the next higher stream than the one on which the message was received. If an integer argument is passed on the command line, the server interprets the argument as the value of `stream_increment`, that is, it decides whether or not to increment the stream number of incoming messages. We will use this option in our discussion of head-of-line blocking in Section 10.5.

Create an SCTP socket

15 An SCTP one-to-many-style socket is created.

Bind an address

16–20 An Internet socket address structure is filled in with the wildcard address (`INADDR_ANY`) and the server's well-known port, `SERV_PORT`. Binding the wildcard address tells the system that this SCTP endpoint will use all available local addresses in any association that is set up. For multihomed hosts, this binding means that a remote endpoint will be able to make associations with and send packets to any of the local host's routeable addresses. Our choice of the SCTP port number is based on Figure 2.10. Note that the server makes the same considerations that were made earlier in our previous example found in Section 5.2.

Set up for notifications of interest

21–23 The server changes its notification subscription for the one-to-many SCTP socket. The server subscribes to just the `sctp_data_io_event`, which will allow the server to see the `sctp_sndrcvinfo` structure. From this structure, the server can determine the stream number on which the message arrived.

Enable incoming associations

24 The server enables incoming associations with the `listen` call. Then, control enters the main processing loop.

Figure 10.2 SCTP streaming echo server.

sctp/sctpserv01.c

```
 1 #include     "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     sock_fd, msg_flags;
 6     char    readbuf [BUFFSIZE];
 7     struct sockaddr_in servaddr, cliaddr;
 8     struct sctp_sndrcvinfo sri;
 9     struct sctp_event_subscribe evnts;
10     int     stream_increment = 1;
11     socklen_t len;
12     size_t rd_sz;

13     if (argc == 2)
14         stream_increment = atoi (argv[1]);
15     sock_fd = Socket (AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);
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     Bind (sock_fd, (SA *) &servaddr, sizeof (servaddr));

21     bzero (&evnts, sizeof (evnts)) ;
```

```
22      evnts.sctp_data_io_event = 1;
23      Setsockopt (sock_fd, IPPROTO_SCTP, SCTP_EVENTS, &evnts, sizeof (evnts)) ;

24      Listen(sock_fd, LISTENQ) ;
25      for ( ; ; ) {
26          len = sizeof(struct sockaddr_in) ;
27          rd_sz = Sctp_recvmsg(sock_fd, readbuf, sizeof (readbuf) ,
28                               (SA *) &cliaddr, &len, &sri, &msg_flags) ;
29          if (stream_increment) {
30              sri.sinfo_stream++;
31              if (sri.sinfo_stream >=
32                 sctp_get_no_strms (sock_fd, (SA *) &cliaddr, len) )
33                 sri.sinfo_stream = 0;
34          }
35          Sctp_sendmsg (sock_fd, readbuf, rd_sz,
36                        (SA *) &cliaddr, len,
37                        sri.sinfo_ppid,
38                        sri.sinfo_flags, sri.sinfo_stream, 0, 0) ;
39      }
40  }
```

Wait for message

26–28 The server initializes the size of the client socket address structure, then blocks while waiting for a message from any remote peer.

Increment stream number if desired

29–34 When a message arrives, the server checks the `stream_increment` flag to see if it should increment the stream number. If the flag is set (no arguments were passed on the command line), the server increments the stream number of the message. If that number grows larger than or equal to the maximum streams, which is obtained by calling our internal function call `sctp_get_no_strms`, the server resets the stream to 0. The function `sctp_get_no_strms` is not shown. It uses the `SCTP_STATUS` SCTP socket option discussed in Section 7.10 to find the number of streams negotiated.

Send back response

35–38 The server sends back the message using the payload protocol ID, flags, and the possibly modified stream number from the `sri` structure.

Notice that this server does not want association notification, so it disables all events that would pass messages up the socket buffer. The server relies on the information in the `sctp_sndrcvinfo` structure and the returned address found in *cliaddr* to locate the peer association and return the echo.

This program runs forever until the user shuts it down with an external signal.