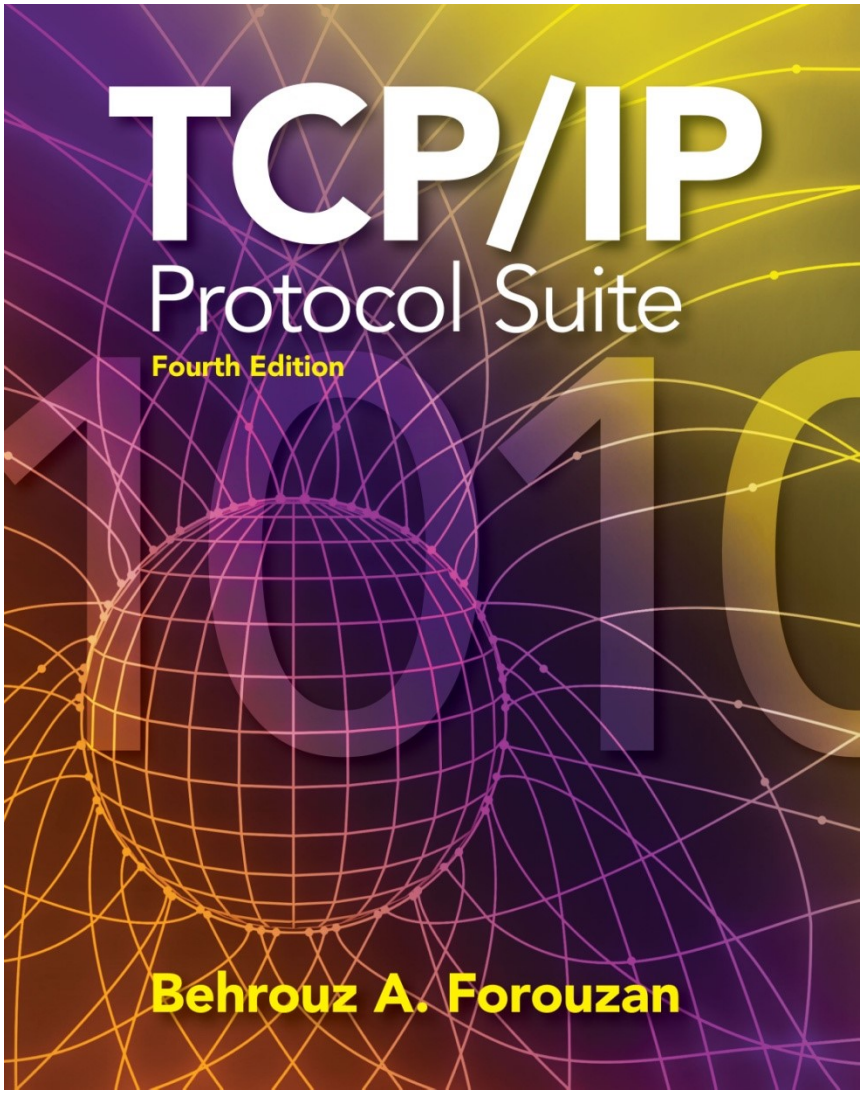
The McGraw-Hill Companies

Chapter 17

Introduction to the Application Layer



OBJECTIVES:

- To introduce client-server paradigm.
- To introduce socket interfaces and list some common functions in this interface.
- To discuss client-server communication using connectionless iterative service offered by UDP.
- To discuss client-server communication using connection-oriented concurrent service offered by TCP.
- To give an example of a client and a server program using UDP.
- To give an example of a client and a server program using TCP.
- To briefly discuss the peer-to-peer paradigm and its application.

Chapter Outline

17-1 CLIENT-SERVER PARADIGM

The purpose of a network, or an internetwork, is to provide services to users: A user at a local site wants to receive a service from a computer at a remote site. One way to achieve this purpose is to run two programs. A local computer runs a program to request a service from a remote computer; the remote computer runs a program to give service to the requesting program. This means that two computers, connected by an internet, must each run a program, one to provide a service and one to request a service.

Topics Discussed in the Section

- ✓ **Server**
- ✓ **Client**
- ✓ **Concurrency**
- ✓ **Socket Interfaces**
- ✓ **Communication Using UDP**
- ✓ **Communication Using TCP**
- ✓ **Predefined Client-Server Applications**

Figure 17.1 *Server types*

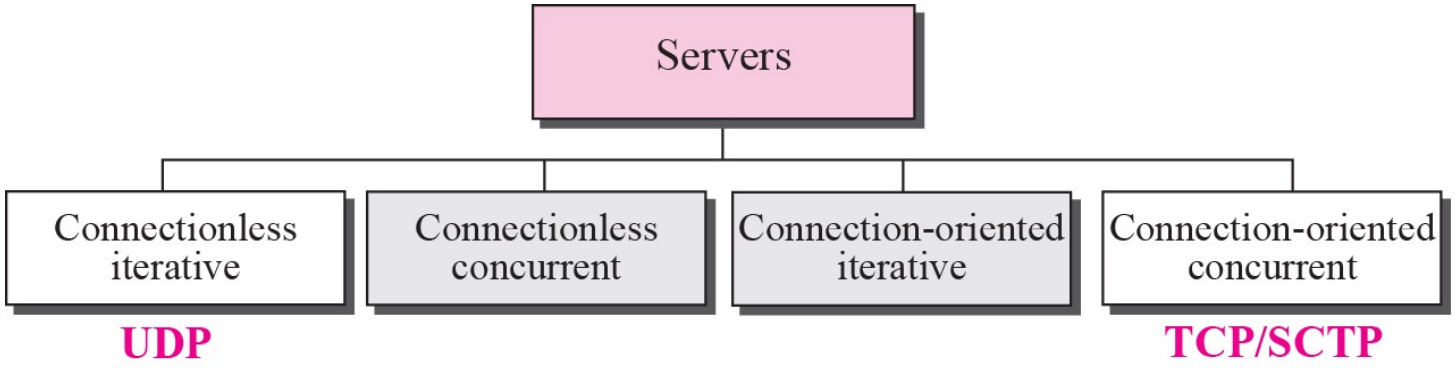


Figure 17.2 *Connectionless iterative server*

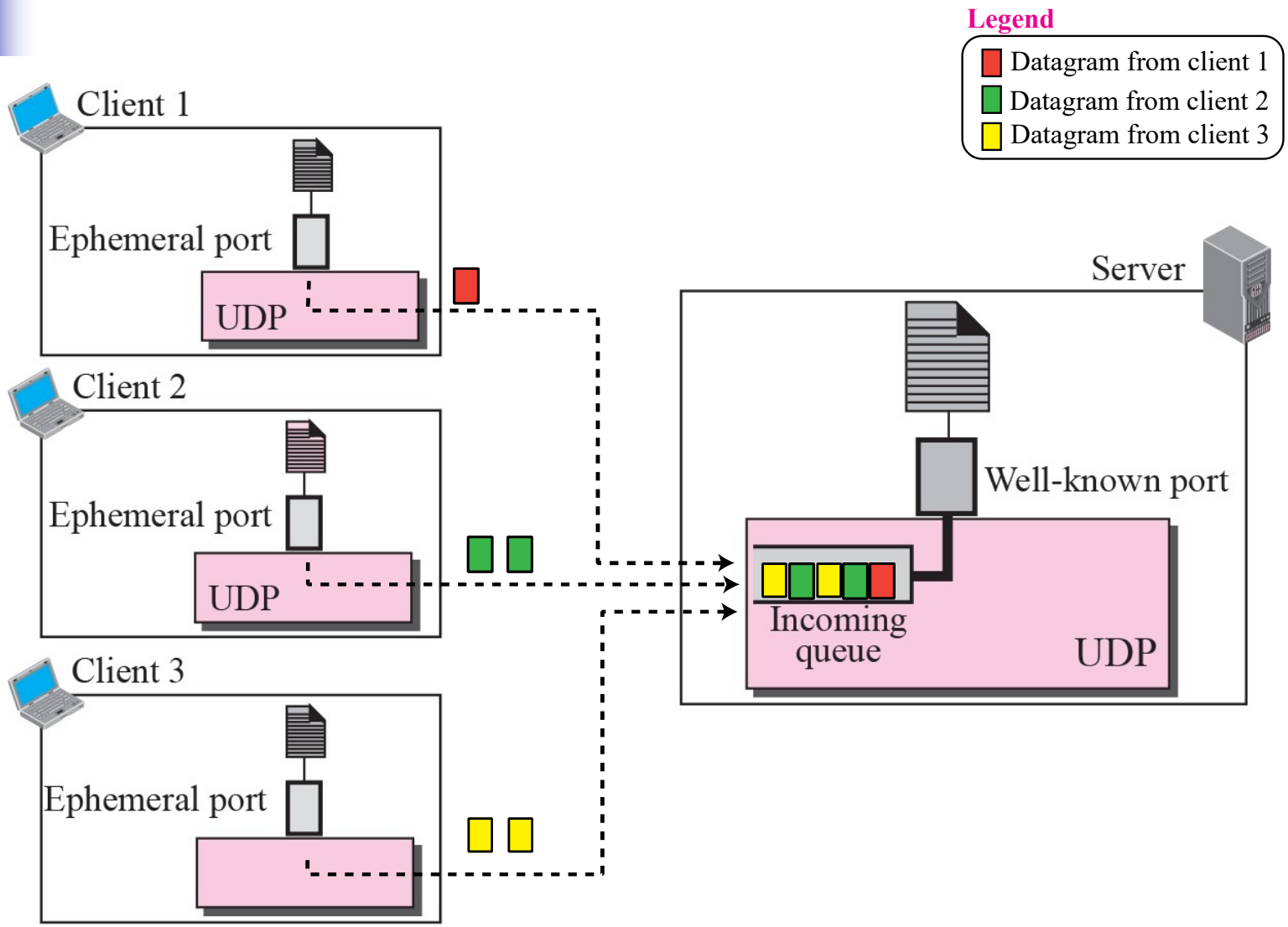
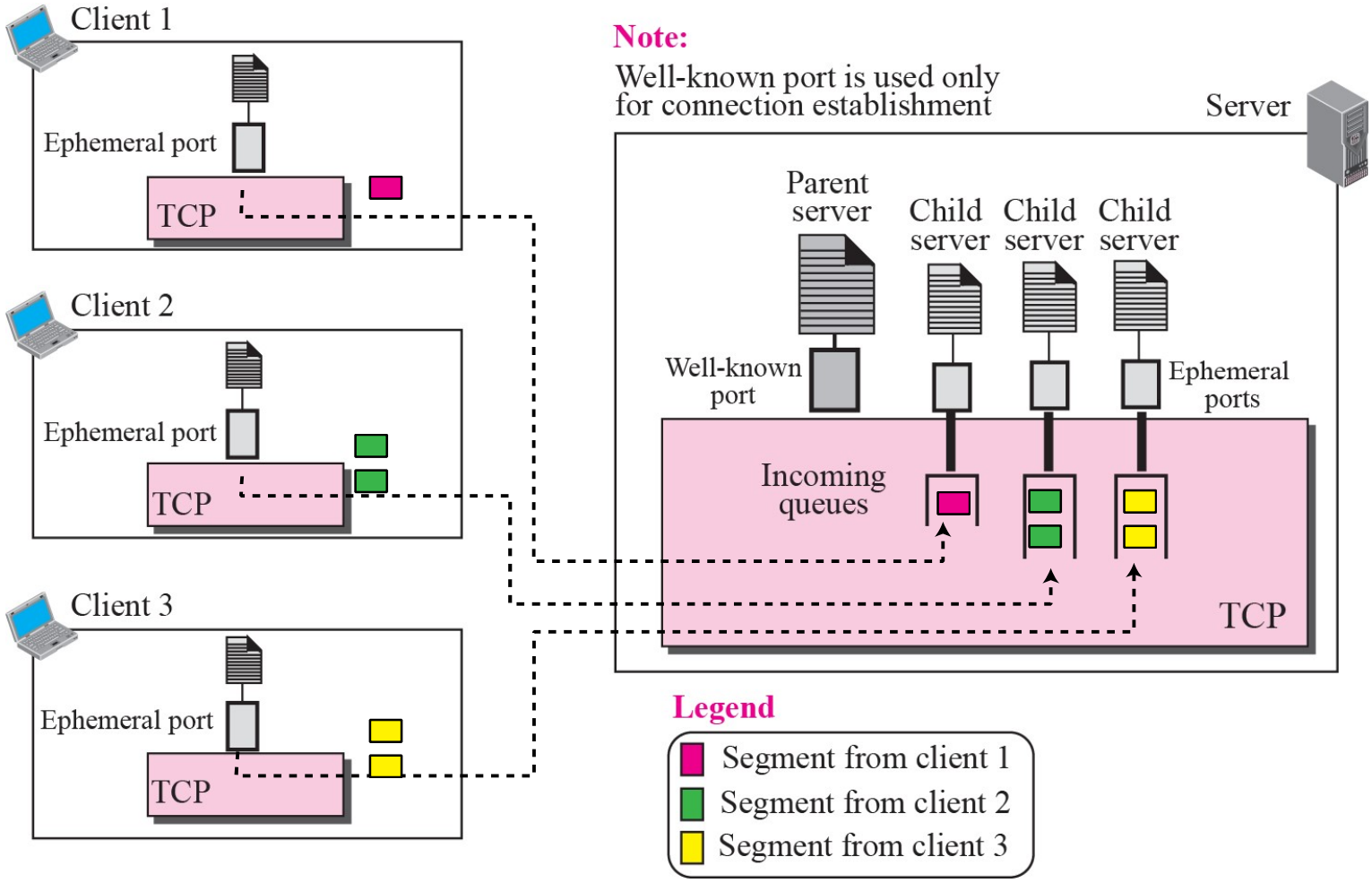


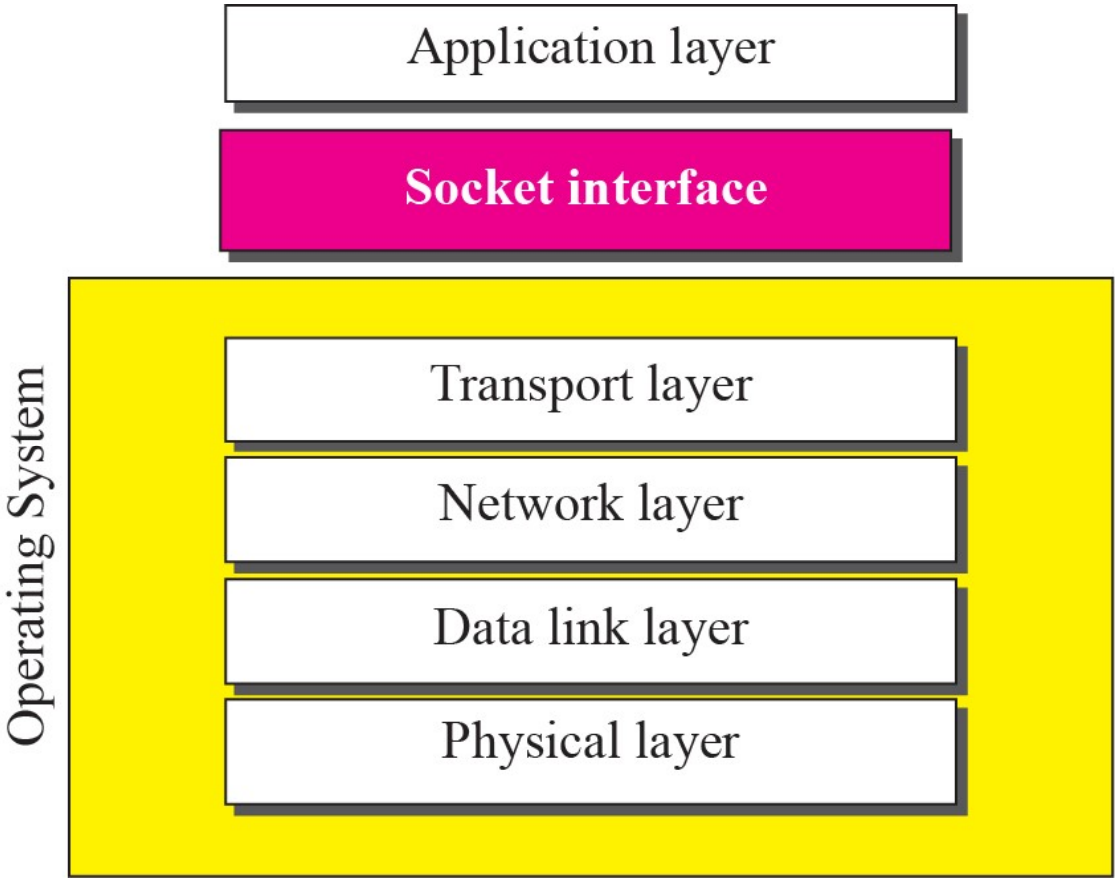
Figure 17.3 *Connection-oriented concurrent server*



Note

An interface is a set of instructions designed for interaction between two entities.

Figure 17.4 *Relation between the operating system and the TCP/IP suite*



Example 17.1

Most of the programming languages have a file interface, a set of instructions that allow the programmer to open a file, read from the file, write to the file, perform other operations on the file, and finally close the file. When a program needs to open the file, it uses the name of the file as it is known to the operation system. When the file is opened, the operating system returns a reference to the file (an integer or pointer) that can be used for other instructions, such as read and write.

Figure 17.5 *Concepts of sockets*

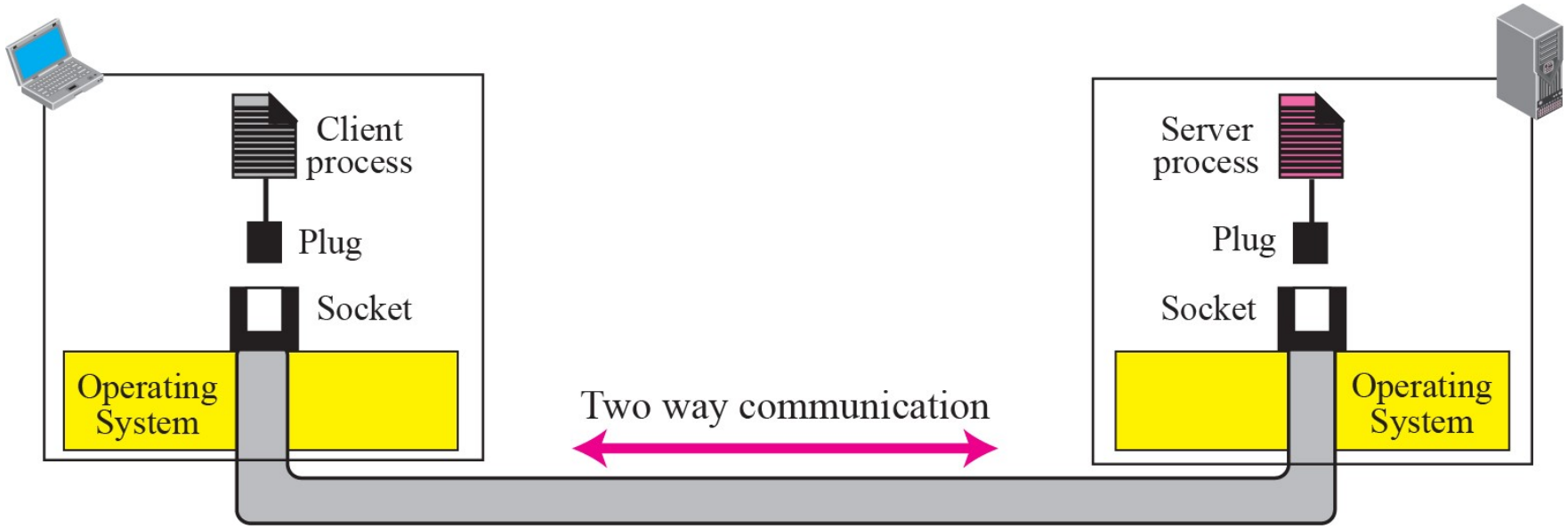
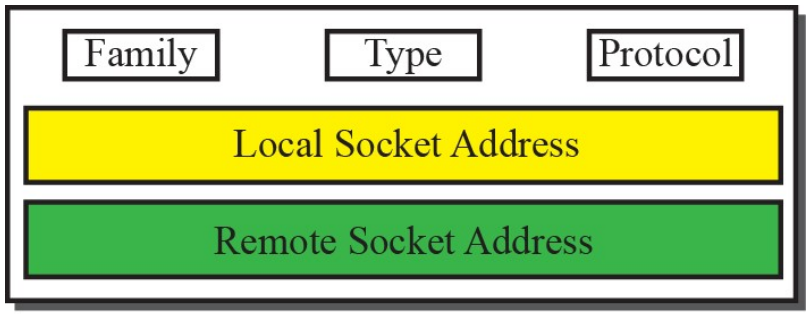


Figure 17.6 *Socket data structure*



Fields

```
struct  socket
{
      int family;
      int type;
      int protocol;
      socketaddr local;
      socketaddr remote;
};
```

Generic definition

Figure 17.7 *Socket types*

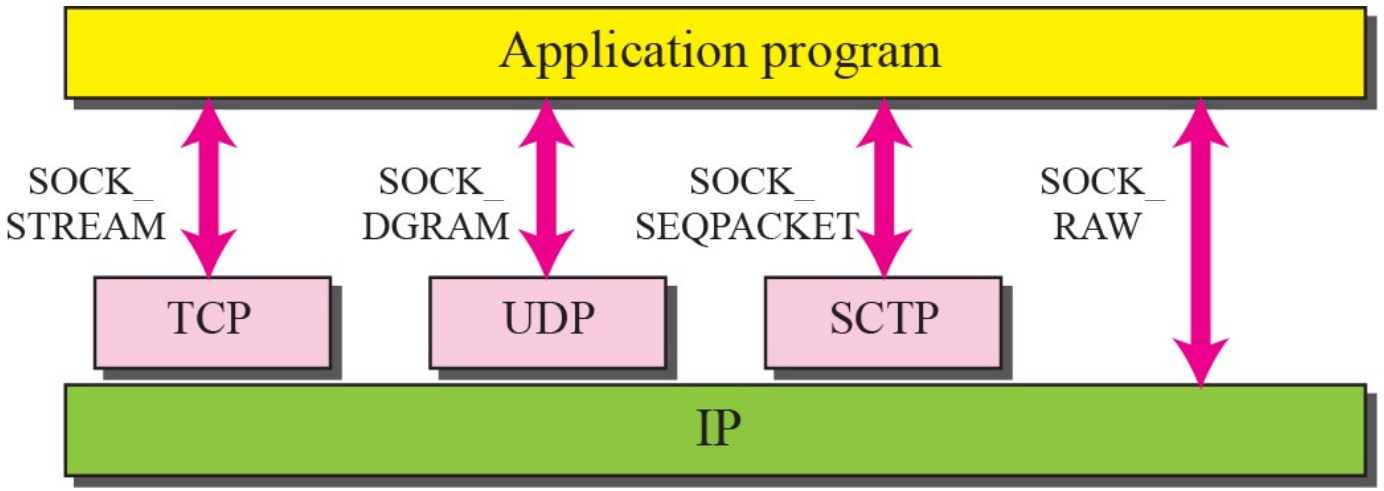


Figure 17.8 *IPv4 socket address*

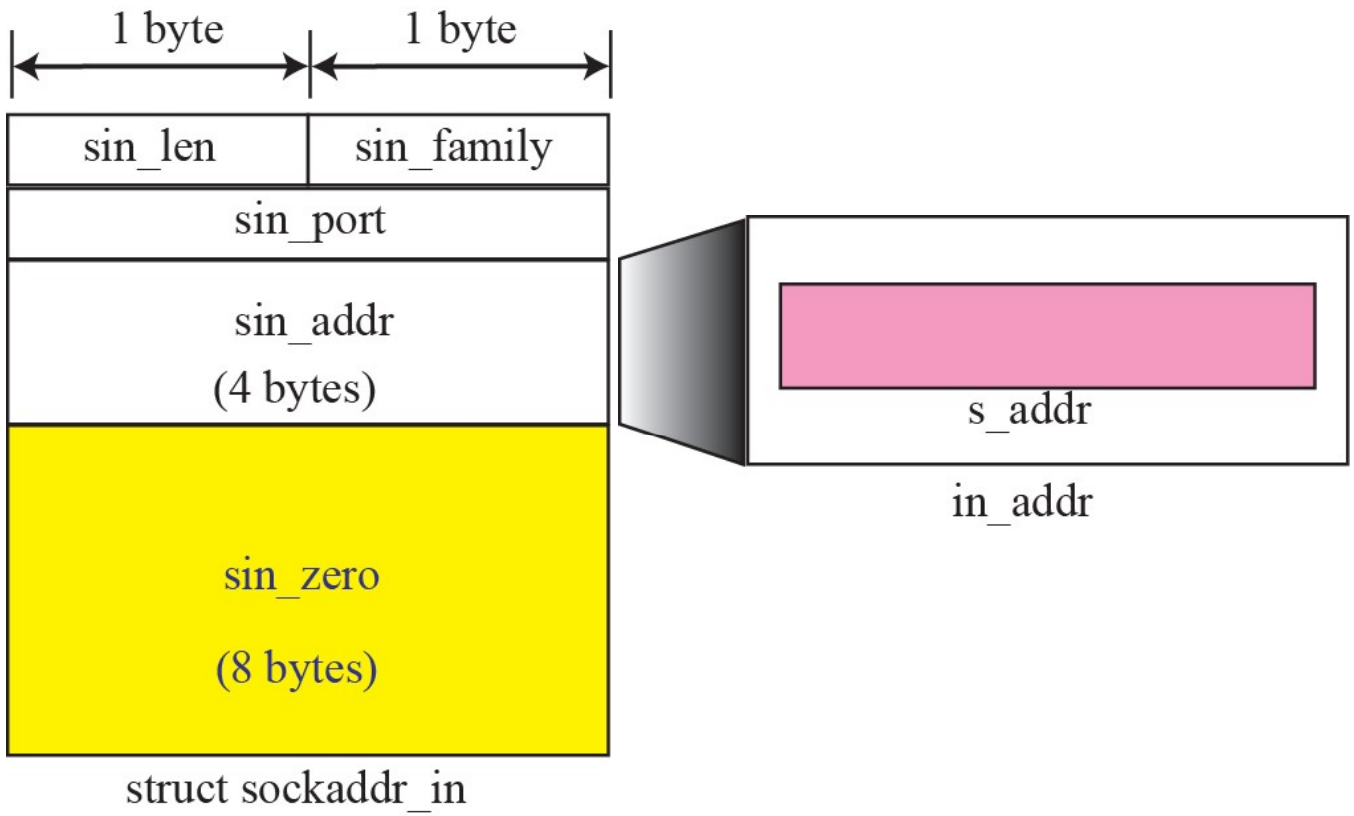
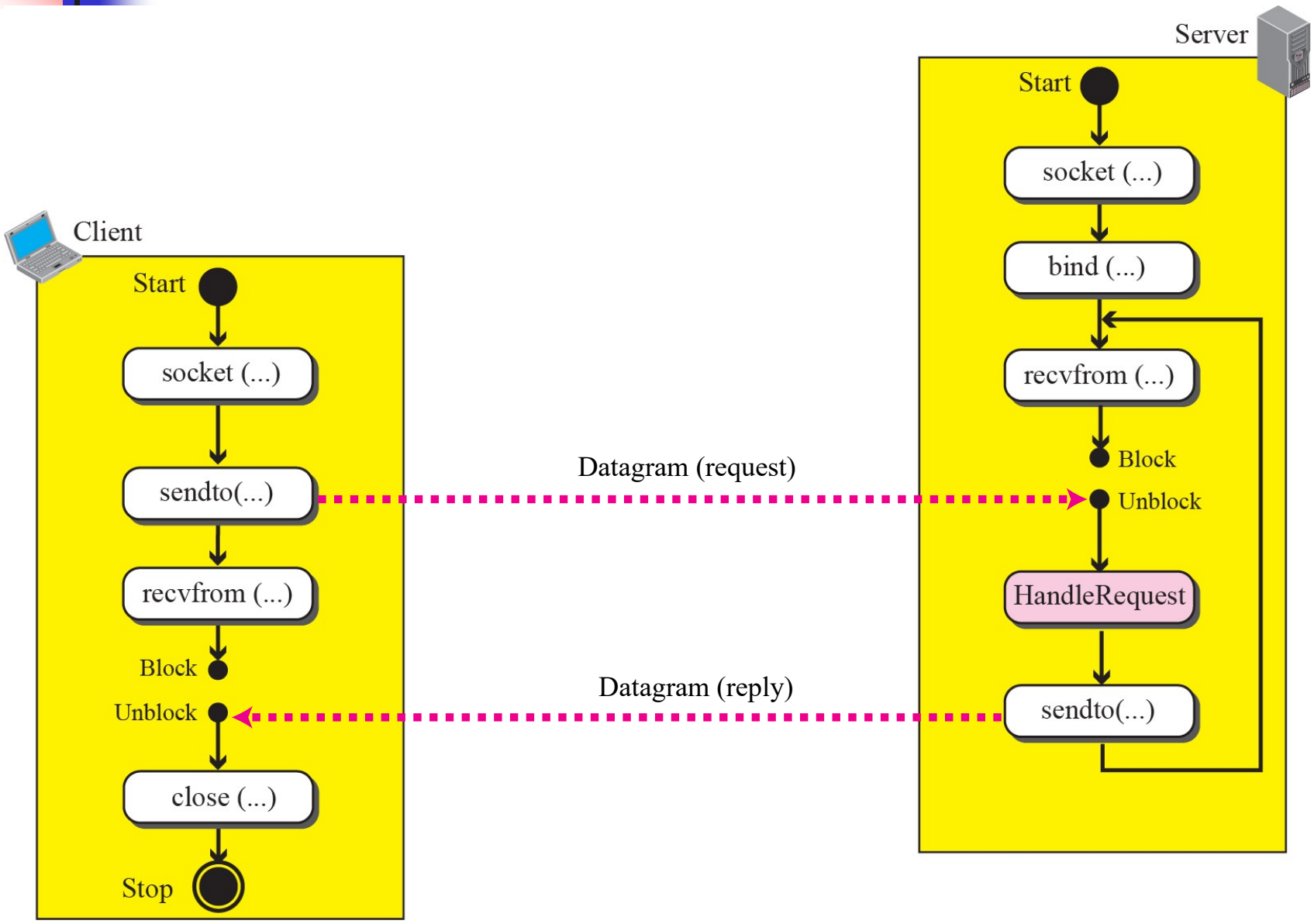


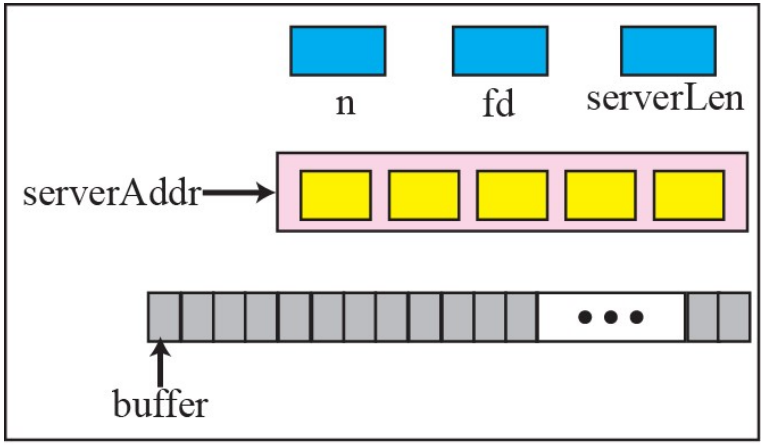
Figure 17.9 *Connectionless iterative communication using UDP*



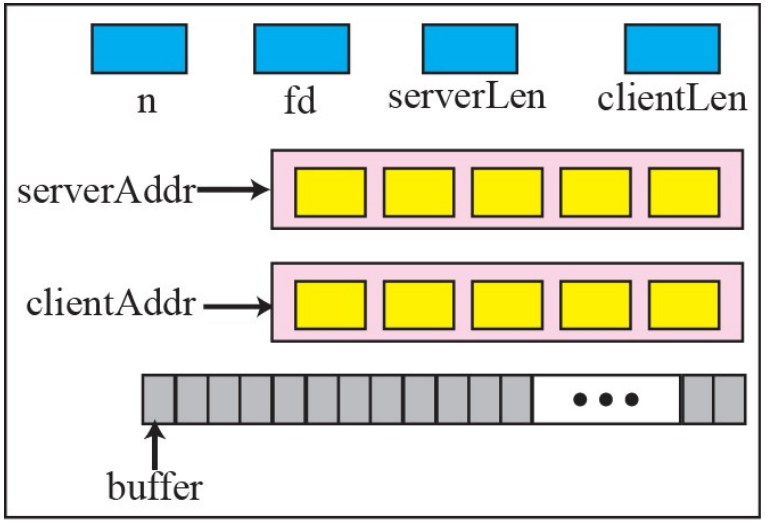
Example 17.2

As an example, let us see how we can design and write two programs: an echo server and an echo server. The client sends a line of text to the server; the server sends the same line back to the client. Although this client/server pair looks useless, it has some applications. It can be used, for example, when a computer wants to test if another computer in the network is alive. To better understand the code in a program, we first give the layout of variables used in both programs as shown in Figure 17.10.

Figure 17.10 *Variables used in echo server and echo client using UDP service*



Variables used by the client process



Variables used by the server process

Table 17.1 *Echo Server Program using the Service of UDP*

```
01 // UDP echo server program
02 #include "headerFiles.h"
03
04 int main (void)
05 {
06       // Declaration and definition
07       int sd;                          // Socket descriptor
08       int nr;                          // Number of bytes received
09       char buffer [256];               // Data buffer
10       struct sockaddr_in serverAddr;   // Server address
11       struct sockaddr_in clientAddr;   // Client address
12       int clAddrLen;                   // Length of client Address
13       // Create socket
14       sd = socket (PF_INET, SOCK_DGRAM, 0);
15       // Bind socket to local address and port
16       memset (&serverAddr, 0, sizeof (serverAddr));
17       serverAddr.sin_family = AF_INET;
```

Table 17.1 *Echo Server Program using the Service of UDP (continued)*

```
18      serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);        // Default address
19      serverAddr.sin_port = htons (7)     // We assume port 7
20      bind (sd, (struct sockaddr*) &serverAddr, sizeof (serverAddr));
21      // Receiving and echoing datagrams
22      for ( ; ; )         // Run forever
23      {
24          nr = recvfrom (sd, buffer, 256, 0, (struct sockaddr*)&clientAddr, &clAddrLen);
25          sendto (sd, buffer, nr, 0, (struct sockaddr*)&clientAddr, sizeof(clientAddr));
26      }
27  } // End of echo server program
```

Table 17.2 *Echo Client Program using the Service of UDP*

```
01  // UDP echo client program
02  #include "headerFiles.h"
04
04  int main (void)
05  {
06      // Declaration and definition
07      int sd;                          // Socket descriptor
08      int ns;                          // Number of bytes send
09      int nr;                          // Number of bytes received
10      char buffer [256];               // Data buffer
11      struct sockaddr_in serverAddr;   // Socket address
11      // Create socket
12      sd = socket (PF_INET, SOCK_DGRAM, 0);
13      // Create server socket address
14      memset (&servAddr, 0, sizeof(serverAddr));
15      servAddr.sin_family = AF_INET;
16      inet_pton (AF_INET, "server address", &serverAddr.sin_addr);
17      serverAddr.sin_port = htons (7);
18      // Send and receive datagrams
19      fgets (buffer, 256, stdin);
```

Table 17.2 *Echo Client Program using the Service of UDP (continued)*

```
20        ns = sendto (sd, buffer, strlen (buffer), 0,
21              (struct sockaddr)&serverAddr, sizeof(serveraddr));
22        recvfrom (sd, buffer, strlen (buffer), 0, NULL, NULL);
23        buffer [nr] = 0;
24        printf ("Received from server:");
25        fputs (buffer, stdout);
26        // Close and exit
27        close (sd);
28        exit (0);
29  } // End of echo client program
```

Note

To be complete, error-checking code needs to be added to both server and client programs.

Figure 17.11 *Flow diagram for connection-oriented, concurrent communication*

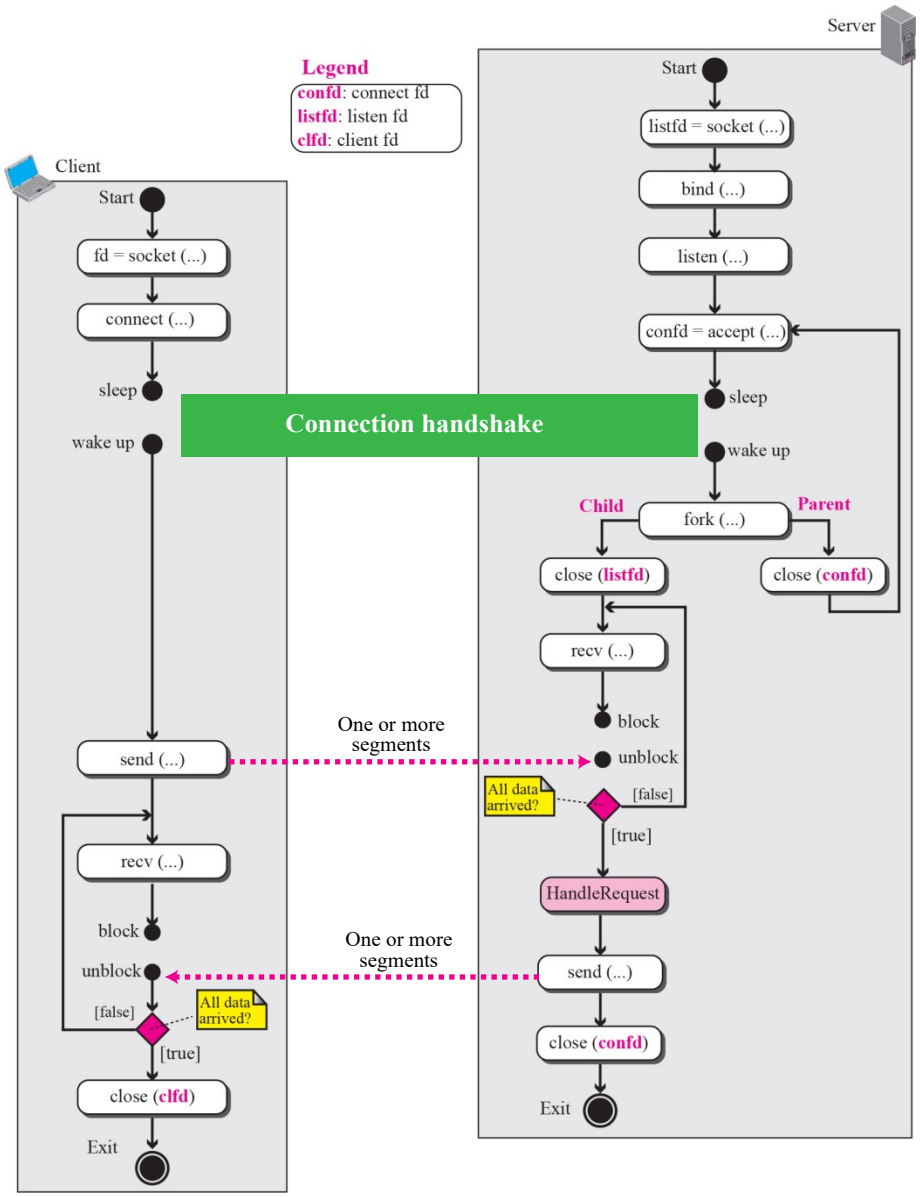
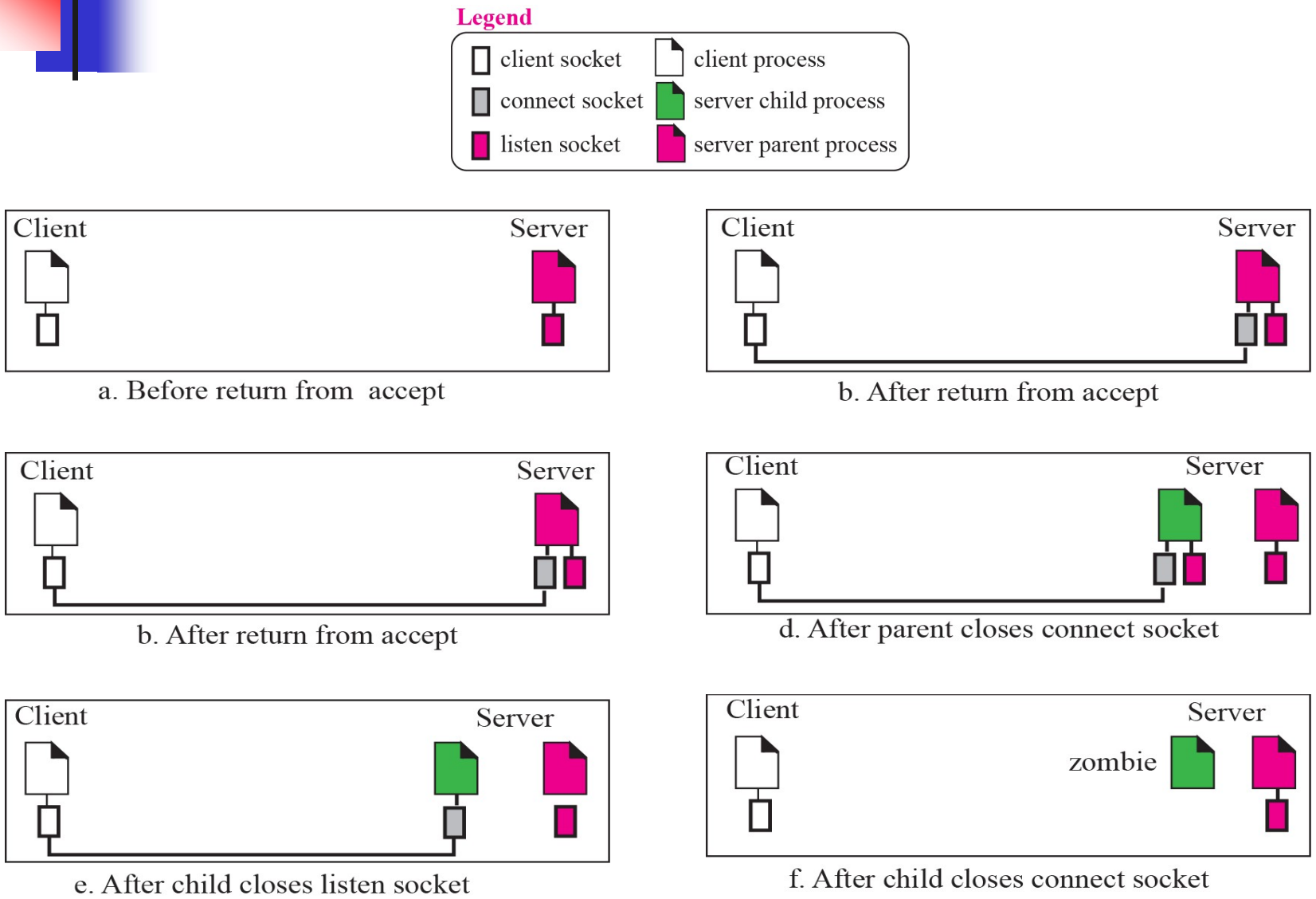


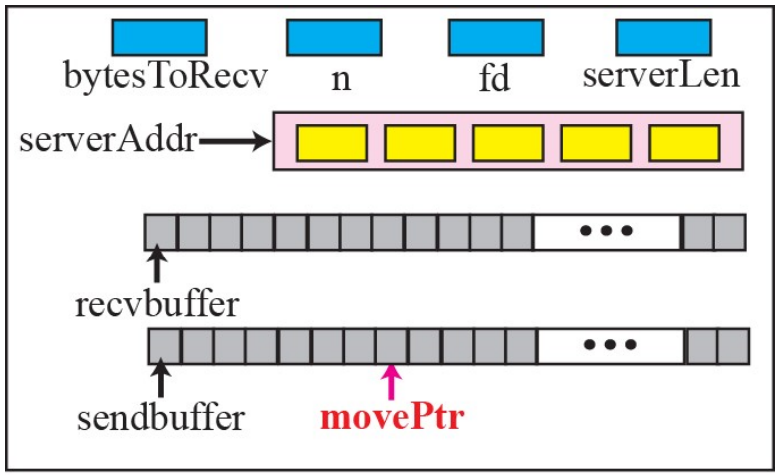
Figure 17.12 *Status of parent and child processes with respect to the socket*



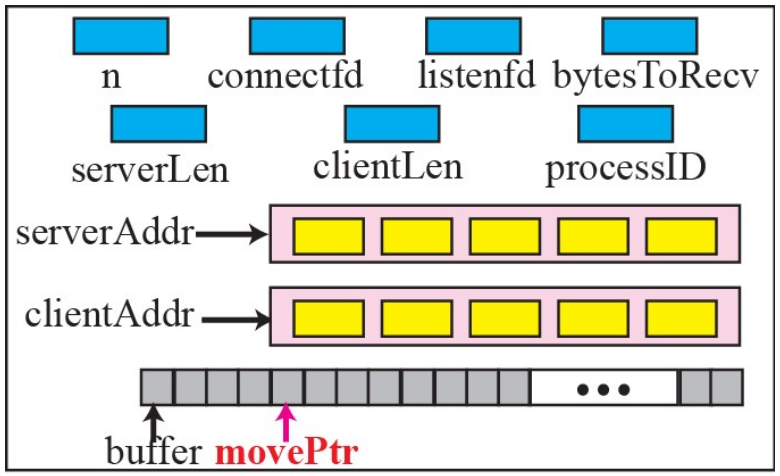
Example 17.3

We want to write two programs to show how we can have an echo client and echo server using the services of TCP. Figure 17.13 shows the variables we use in these two programs. Since data may arrive in different chunks, we need pointers to point to the buffer. The first buffer is fixed and always points to the beginning of the buffer; the second pointer is moving to let the arrived bytes be appended to the end of the previous section.

Figure 17.13 *Variable used in echo client and echo sever using TCP*



Variables used by the client process



Variables used by the server process

Table 17.3 *Echo Server Program using the Services of TCP*

```
01 // Echo server program
02 #include "headerFiles.h"
03
04 int main (void)
05 {
06     // Declaration and definition
07     int listensd;                          // Listen socket descriptor
08     int connectsd;                         // Connecting socket descriptor
09     int n;                                 // Number of bytes in each reception
10     int  bytesToRecv;                      // Total bytes to receive
11     int   processID;                       // ID of the child process
12     char buffer [256];                     // Data buffer
13     char* movePtr;                         // Pointer to the buffer
14     struct sockaddr_in serverAddr;         // Server address
15     struct sockaddr_in clientAddr;         // Client address
16      int clAddrLen;                        // Length of client address
17     // Create listen socket
18     listensd = socket (PF_INET, SOCK_STREAM, 0);
19     // Bind listen socket to the local address and port
20     memset (&serverAddr, 0, sizeof (serverAddr));
21     serverAddr.sin_family = AF_INET;
```

Table 17.3 *Echo Server Program using the Services of TCP (continued)*

```
22      serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);
23      serverAddr.sin_port = htons (7);  // We assume port 7
24      bind (listensd, &serverAddr, sizeof (serverAddr));
25      // Listen to connection requests
26       listen (listensd, 5);
27      // Handle the connection
28       for ( ; ; )                    // Run forever
29      {
30            connectsd = accept (listensd, &clientAddr, &clAddrLen);
31            processID = fork ();
32            if (processID == 0)                    // Child process
33            {
34                  close (listensd);
35                  bytesToRecv = 256;
36                  movePtr = buffer;
37                  while ( (n = recv (connectfd, movePtr, bytesToRecv, 0)) > 0)
38                  {
39                        movePtr = movePtr + n;
40                        bytesToRecv = movePtr – n;
41                  }  // End of while
```

Table 17.3 *Echo Server Program using the Services of TCP (continued)*

```
42                  send (connectsd, buffer, 256, 0);
43                  exit (0);
44              } // End of if
45              close (connectsd):                    // Back to parent process
46          } // End of for loop
47  } // End of echo server program
```

Table 17.4 *Echo Client Program using the services of TCP*

```
01 // TCP echo client program
02 #include "headerFiles.h"
03
04 int main (void)
05 {
06     // Declaration and definition
07     int  sd;                          // Socket descriptor
08     int  n;                           // Number of bytes received
09     int  bytesToRecv;                 // Number of bytes to receive
10     char  sendBuffer [256];           // Send buffer
11     char  recvBuffer [256];           // Receive buffer
12     char* movePtr;                    // A pointer the received buffer
13     struct sockaddr_in  serverAddr;   // Server address
14
15     // Create socket
16     sd = socket (PF_INET, SOCK_STREAM, 0);
17     // Create server socket address
18     memset (&serverAddr, 0, sizeof(serverAddr);
19     serverAddr.sin_family = AF_INET;
20     inet_pton (AF_INET, "server address", &serverAddr.sin_addr);
21     serverAddr.sin_port = htons (7);    // We assume port 7
22     // Connect
```

Table 17.4 *Echo Client Program using the services of TCP (continued)*

```
23      connect (sd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
24      // Send and receive data
25      fgets (sendBuffer, 256, stdin);
26      send (fd, sendBuffer, strlen (sendbuffer), 0);
27      bytesToRecv = strlen (sendbuffer);
28      movePtr = recvBuffer;
29      while ( (n = recv (sd, movePtr, bytesToRecv, 0) ) > 0)
30      {
31          movePtr = movePtr + n;
32          bytesToRecv = bytesToRecv – n;
33      } // End of while loop
34      recvBuffer[bytesToRecv] = 0;
35      printf ("Received from server:");
36      fputs (recvBuffer, stdout);
37      // Close and exit
38      close (sd);
39      exit (0);
40  } // End of echo client program
```

Note

In Appendix F we give some simple Java versions of programs in Table 17.1 to 17.4

17-2 PEER-TO-PEER PARADIGM

Although most of the applications available in the Internet today use the client-server paradigm, the idea of using peer-to-peer (P2P) paradigm recently has attracted some attention. In this paradigm, two peer computers can communicate with each other to exchange services. This paradigm is interesting in some areas such file as transfer in which the client-server paradigm may put a lot of the load on the server machine. However, we need to mention that the P2P paradigm does not ignore the client-server paradigm; it is based on this paradigm.