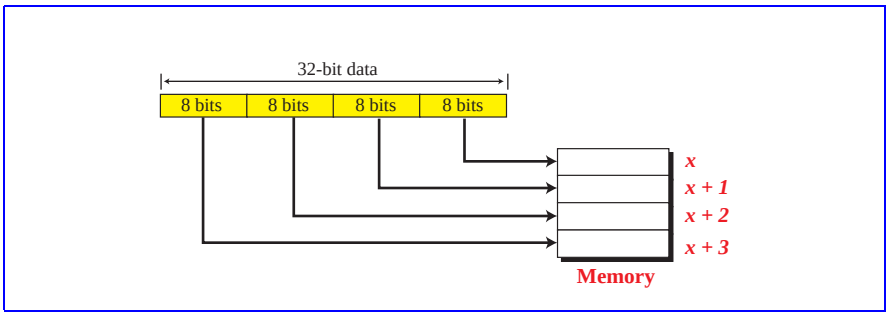
CHAPTER 17

Introduction to the Application Layer

Exercise

1. See Figure 17.E1. The rightmost (*little*) byte of a 32-bit integer is stored in the first memory location (*x*).

Figure 17.E1 *Solution to Exercise 1*



3. A union can be used to define two alternative data types or a combination of a data type and a data structure. We define a union as a combination of a 16-bit integer and an array of two characters. We stored an integer in the union, but use the union as an array of two bytes as shown in the program in Table 17.E3.

Table 17.E3 *Solution to Exercise 3*

```
01  // A program to test the computer byte-order
02  #include "headerFiles.h"
03  union
04  {
05          uint16-t    intData;
06          char    charData[2];
07  }data;
09  int main (void)
10  {
11          data.intData = 0x0102;
```

Table 17.E3 *Solution to Exercise 3*

```
12          if (data.charData[0] == 1 && (data.charData[1] == 2)
13                  printf ("Big-endian byte order\n");
14          else if (data.charData[0] == 2 && (data.charData[1] == 1)
15                  printf ("Little-endian byte order\n");
16          else
17                  printf ("Unknown byte order\n");
18  } //End of main
```

5. Table 17.E5 shows a simple program. In lines 17 and18, we create two chunks of memory using the ***memset*** function and store the string "AAAAAAAAA" in the first and the string "BBBBBBBBBBB" in the second. In line 21, we use the function **memcpy** to copy only the first five characters of the second chunk into the first. In line 23, we use the ***memcmp*** function to compare first the first bytes of the two chunks. In line 24, we use the ***memcmp*** to compare the six bytes of the two chunks.

Table 17.E5 *Solution to Exercise 5*

```
01  // A program to test the computer byte-order
02  #include "headerFiles.h"
03
04  int main (void)
05  {
06          // Declaration and definition
07          char buff1 [10];
08          char buff2 [12];
09          int res1;
10          int res2;
11          // Statements
12          memset (buff1, 'A', 10);
13          memset (buff2, 'B', 12);
14          printf ("%s\n", buff1);
15          printf ("%s\n", buff2);
16          memcpy (buff1, buff2, 5);
17          printf ("%s\n", buff1);
18          res1 = memcmp (buff1, buff2, 5);
19          res2 = memcmp (buff1, buff2, 6);
20          printf ("%d %d \n", res1, res2);
21          return 0;
22  } //End of main
```

The result of running the program is shown below. The first line shows the contents of the first chunk. The second line shows the contents of the second chunk. the third line shows the contents the second chunk after copying. The last line shows the result of the comparison.

```
AAAAAAAAAA
BBBBBBBBBBBB
```

```
BBBBBAAAAA
0 -1
```

7. We can replace line 14 in Table 17.1 in the text with the following lines:

```
if (sd = socket (PF_INET, SOCK_DGRAM, 0) = –1)
{
    perror("Call to socket function failed”);
    exit (1);
}
```

9. Table 17.E9 shows the new program. The colored section (lines 21 to 28) shows the changes in the original program in the text.

Table 17.E9 *Solution to Exercise 9*

```
01  // UDP echo client program
02  #include "headerFiles.h"
03
04  int main (void)
05  {
06      // Declaration and definition
07      int sd;
08      int ns;
09      int nr;
10      char buffer[256];
11      struct   sockaddr_in    serverAddr;
12
13      // Create socket
14      sd = socket (PF_INET, SOCK_DGRAM, 0);
15
16      // Create server socket address
17      memset (&servAddr, 0, sizeof(serverAddr));
18      servAddr.sin_family = AF_INET;
19      inet_pton (AF_INET, “server address”, &serverAddr.sin_addr);
20      serverAddr.sin_port = htons (7);
21
22      // Send and receive datagrams
23      while (fgets (buffer, 256, stdin));
24      {
25          ns = sendto (sd, buffer, strlen (buffer), 0,
26              (struct sockaddr)&serverAddr, sizeof(serveraddr));
27          recvfrom (sd, buffer, strlen (buffer), 0, NULL, NULL);
28          buffer [nr] = 0;
29          printf (“Received from server:”);
30          fputs (buffer, stdout);
31      }
32
```

Table 17.E9 *Solution to Exercise 9 (Continued)*

```
33          // Close and exit
34          close (sd);
35          exit (0);
36
37      } // End of echo client program
```

10. See Figure 17.E10.

Figure 17.E10 *Solution to Exercise 10*