As you read from Section 3.2 of TCP/IP Sockets in C: Practical Guide for Programmers, different architectures use different byte ordering for multibyte quantities. A big-endian machine places the most significant byte in the lowest address while a little-endian machine places the least significant byte in the lowest address. To avoid confusion when communicating between different architectures, the Sockets interface specifies a standard byte ordering called network-byte order, which happens to be big-endian. Consequently, all network communication should be big-endian, irrespective of the client or server architecture. Sockets provides a set of macros to convert to and from host to network byte order (i.e., [hn]to[nh][sl]( )).

Consider the following C program:

```c
#include <stdio.h>

main() {
    int i; /* Loop variable */
    long x = 0x112A380; /* Value to play with */
    unsigned char *ptr = (char *) &x; /* Byte pointer */

    /* Observe value in host byte order */
    printf("x in hex: %x\n", x);
    printf("x by bytes: ");
    for (i=0; i < sizeof(long); i++)
        printf("%x\t", ptr[i]);
    printf("\n");

    /* Observe value in network byte order */
    x = htonl(x);
    printf("\nAfter htonl()\n"lyer byte order */
    printf("x in hex: %x\n", x);
    printf("x by bytes: ");
    for (i=0; i < sizeof(long); i++)
        printf("%x\t", ptr[i]);
    printf("\n");
}
```

This program shows how the long variable `x` with value 112A380 (hexadecimal) is stored. When this program is executed on a little-endian processor, it outputs the following:

```
x in hex: 112a380
x by bytes: 80  a3   12   1

After htonl()
x in hex: 80a31201
x by bytes: 1    12   a3   80
```
When we examine the individual bytes of \( x \), we find the least significant byte (0x80) in the lowest address. After we call htonl() to convert to network byte order, we get the most significant byte (0x1) in the lowest address. Of course, if we try to print the value of \( x \) after converting its byte order, we get a meaningless number.

Let’s execute the same program on a big-endian processor:

\[
\begin{align*}
\text{x in hex: } & \quad 112a380 \\
\text{x by bytes: } & \quad 1 \quad 12 \quad a3 \quad 80 \\
\end{align*}
\]

After htonl()
\[
\begin{align*}
\text{x in hex: } & \quad 112a380 \\
\text{x by bytes: } & \quad 1 \quad 12 \quad a3 \quad 80 \\
\end{align*}
\]

Here we find the most significant byte (0x1) in the lowest address. Calling htonl() to convert to network byte order does not change anything because network byte order is already big endian.