```
/*****************************************************************/
/* This sample program provides a code for a connectionless client.       */
/*****************************************************************/

/*****************************************************************/
/* Header files needed for this sample program                           */
/*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

/*****************************************************************/
/* Constants used by this program                                        */
/*****************************************************************/
#define SERVER_PORT     3555
#define BUFFER_LENGTH    100
#define FALSE              0
#define SERVER_NAME     "ServerHostName"

/* Pass in 1 parameter which is either the */
/* address or host name of the server, or  */
/* set the server name in the #define      */
/* SERVER_NAME                             */
void main(int argc, char *argv[])
{
    /*****************************************************************/
    /* Variable and structure definitions.                           */
    /*****************************************************************/
    int    sd, rc;
    char   server[NETDB_MAX_HOST_NAME_LENGTH];
    char   buffer[BUFFER_LENGTH];
    struct sockaddr_in6 serveraddr;
    int    serveraddrlen = sizeof(serveraddr);
    struct addrinfo hints, *res;

    /*****************************************************************/
    /* A do/while(FALSE) loop is used to make error cleanup easier.  The   */
    /* close() of the socket descriptor is only done once at the very end  */
    /* of the program.                                               */
    /*****************************************************************/
    do
    {
        /*************************************************************/
        /* The socket() function returns a socket descriptor, representing  */
        /* an endpoint.  The statement also identifies that the INET6       */
        /* (Internet Protocol) address family with the UDP transport        */
        /* (SOCK_DGRAM) will be used for this socket.                       */
        /*************************************************************/
        sd = socket(AF_INET6, SOCK_DGRAM, 0);
        if (sd < 0)
        {
            perror("socket() failed");
```

```
    break;
}

/*****************************************************************/
/* If an argument was passed in, use this as the server, otherwise  */
/* use the #define that is located at the top of this program.      */
/*****************************************************************/
if (argc > 1)
    strcpy(server, argv[1]);
else
    strcpy(server, SERVER_NAME);

memset(&serveraddr, 0, sizeof(serveraddr));
serveraddr.sin6_family      = AF_INET6;
serveraddr.sin6_port        = htons(SERVER_PORT);
rc = inet_pton(AF_INET6, server, &serveraddr.sin6_addr.s6_addr);
if (rc != 1)
{
    /*****************************************************************/
    /* The server string that was passed into the inet_pton()        */
    /* function was not a hexidecimal colon IP address.  It must     */
    /* therefore be the hostname of the server.  Use the             */
    /* getaddrinfo() function to retrieve the IP address of the      */
    /* server.                                                       */
    /*****************************************************************/
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET6;
    hints.ai_flags = AI_V4MAPPED;
    rc = getaddrinfo(server, NULL, &hints, &res);
    if (rc != 0)
    {
        printf("Host not found! (%s)", server);
        break;
    }

    memcpy(&serveraddr.sin6_addr,
           (&((struct sockaddr_in6 *)(res->ai_addr))->sin6_addr),
           sizeof(serveraddr.sin6_addr));

    freeaddrinfo(res);
}

/*****************************************************************/
/* Initialize the data block that is going to be sent to the server */
/*****************************************************************/
memset(buffer, 0, sizeof(buffer));
strcpy(buffer, "A CLIENT REQUEST");

/*****************************************************************/
/* Use the sendto() function to send the data to the server.        */
/*****************************************************************/
rc = sendto(sd, buffer, sizeof(buffer), 0,
            (struct sockaddr *)&serveraddr,
            sizeof(serveraddr));
if (rc < 0)
```

```
        {
            perror("sendto() failed");
            break;
        }

        /*****************************************************************/
        /* Use the recvfrom() function to receive the data back from the    */
        /* server.                                                       */
        /*****************************************************************/
        rc = recvfrom(sd, buffer, sizeof(buffer), 0,
                      (struct sockaddr *)&serveraddr,
                      & serveraddrlen);
        if (rc < 0)
        {
            perror("recvfrom() failed");
            break;
        }

        printf("client received the following: <%s>\n", buffer);
        inet_ntop(AF_INET6, &serveraddr.sin6_addr.s6_addr,
                  buffer, sizeof(buffer));
        printf("from port %d, from address %s\n",
               ntohs(serveraddr_sin6_port), buffer);

        /*****************************************************************/
        /* Program complete                                              */
        /*****************************************************************/

    } while (FALSE);

    /*****************************************************************/
    /* Close down any open socket descriptors                        */
    /*****************************************************************/
    if (sd != -1)
        close(sd);
}
```