```
/*****************************************************************/
/* This is an IPv4 or IPv6 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 BUFFER_LENGTH    250
#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=-1, rc, bytesReceived=0;
  char   buffer[BUFFER_LENGTH];
  char   server[NETDB_MAX_HOST_NAME_LENGTH];
  char   servport[] = "3005";
  struct in6_addr serveraddr;
  struct addrinfo hints, *res=NULL;

  /*********************************************************************/
  /* 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 along with the free of the list of addresses.        */
/*********************************************************************/
  do
  {
    /*******************************************************************/
    /* 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(&hints, 0x00, sizeof(hints));
    hints.ai_flags    = AI_NUMERICSERV;
    hints.ai_family   = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    /*******************************************************************/
    /* Check if we were provided the address of the server using        */
```

```
/* inet_pton() to convert the text form of the address to binary    */
/* form. If it is numeric then we want to prevent getaddrinfo()    */
/* from doing any name resolution.                                 */
/*******************************************************************/
rc = inet_pton(AF_INET, server, &serveraddr);
if (rc == 1)    /* valid IPv4 text address? */
{
  hints.ai_family = AF_INET;
  hints.ai_flags |= AI_NUMERICHOST;
}
else
{
  rc = inet_pton(AF_INET6, server, &serveraddr);
  if (rc == 1) /* valid IPv6 text address? */
  {

    hints.ai_family = AF_INET6;
    hints.ai_flags |= AI_NUMERICHOST;
  }
}
/*******************************************************************/
/* Get the address information for the server using getaddrinfo().  */
/*******************************************************************/
rc = getaddrinfo(server, servport, &hints, &res);
if (rc != 0)
{
  printf("Host not found --> %s\n", gai_strerror(rc));
  if (rc == EAI_SYSTEM)
    perror("getaddrinfo() failed");
  break;
```

```
}

/*****************************************************************/
/* The socket() function returns a socket descriptor, which represents   */
/* an endpoint.  The statement also identifies the address family,  */
/* socket type, and protocol using the information returned from    */
/* getaddrinfo().                                                */
/*****************************************************************/
sd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
if (sd < 0)
{
   perror("socket() failed");
   break;
}
/*****************************************************************/
/* Use the connect() function to establish a connection to the      */
/* server.                                                       */
/*****************************************************************/
rc = connect(sd, res->ai_addr, res->ai_addrlen);
if (rc < 0)
{
   /*************************************************************/
   /* Note: the res is a linked list of addresses found for server. */
   /* If the connect() fails to the first one, subsequent addresses */
   /* (if any) in the list can be tried if required.              */
   /*************************************************************/
   perror("connect() failed");
   break;
}
```

```
/*********************************************************************/
/* Send 250 bytes of a's to the server                               */
/*********************************************************************/
memset(buffer, 'a', sizeof(buffer));
rc = send(sd, buffer, sizeof(buffer), 0);
if (rc < 0)
{
  perror("send() failed");
  break;
}

/*********************************************************************/
/* In this example we know that the server is going to respond with */
/* the same 250 bytes that we just sent.  Since we know that 250    */
/* bytes are going to be sent back to us, we can use the            */
/* SO_RCVLOWAT socket option and then issue a single recv() and     */
/* retrieve all of the data.                                        */
/*                                                                  */
/* The use of SO_RCVLOWAT is already illustrated in the server      */
/* side of this example, so we will do something different here.    */
/* The 250 bytes of the data may arrive in separate packets,        */
/* therefore we will issue recv() over and over again until all     */
/* 250 bytes have arrived.                                          */
/*********************************************************************/
while (bytesReceived < BUFFER_LENGTH)
{
  rc = recv(sd, & buffer[bytesReceived],
            BUFFER_LENGTH - bytesReceived, 0);
  if (rc < 0)
  {
```

```
        perror("recv() failed");
        break;
      }
      else if (rc == 0)
      {
        printf("The server closed the connection\n");
        break;
      }

      /*********************************************************/
      /* Increment the number of bytes that have been received so far  */
      /*********************************************************/
      bytesReceived += rc;
    }

  } while (FALSE);

  /*******************************************************************/
  /* Close down any open socket descriptors                          */
  /*******************************************************************/
  if (sd != -1)
    close(sd);
  /*******************************************************************/
  /* Free any results returned from getaddrinfo                      */
  /*******************************************************************/
  if (res != NULL)
    freeaddrinfo(res);
}
```