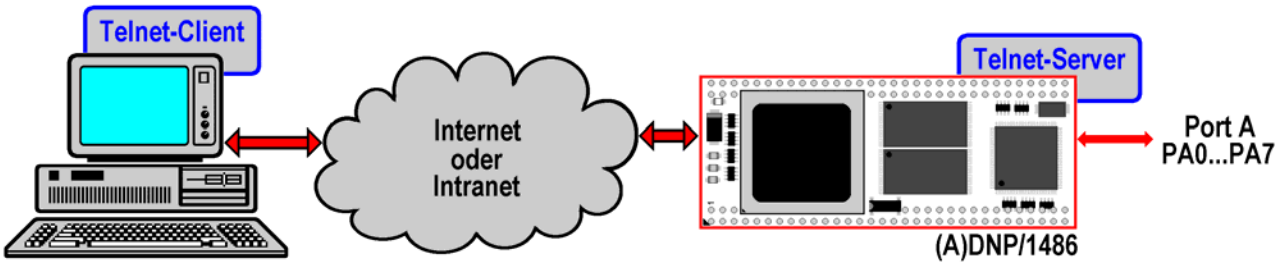
How to write a Measurement Telnet Server

A measurement Telnet server allows you to access remote I/Os with a standard Telnet client program. The following samples shows a way to set the LEDs of a DNP/EVA1 or 2 evaluation board with a Telnet client program.



The C sample code in this document forms a measurement (MSR) Telnet server for the (A)DNP/1486. The server allows you to enter values between 0 and 255 within a Telnet client window. This value is converted to a 8-bit binary number. The measurement Telnet server writes the binary number to the (A)DNP/1486 parallel I/O port A. This port drives the DNP/EVA1 or 2 LED array.

- **1. Step:** Download the sample source code **`msrtsvr2.c`** from **www.dilnetpc.com** (i.e. enter the URL **`www.dilnetpc.com/msrtsvr2.c`** to your browser). The following listing shows this sample code. Please note: We have remove some text within some **`printf`**-statements for cutting the total length of some lines.

```
// msrtsvr2.c: Simple MSR Telnet server program. Use gcc for x86 …
// Vers. 2.00 - 24.July 2002
// k.d.walter@t-online.de

// Includes

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <asm/io.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

// Defines

#define  CSCIR  0x22      // SC410: Chip Setup and Control Index Register
#define  CSCDR  0x23      // SC410: Chip Setup and Control Data Register
#define   PAMR   0xa5     // SC410: PIO Port A Mode Register
```

```
#define  PADR   0xa9     // SC410: PIO Port A Data Register

// Functions

int main (int argc, char* argv[])
{
   int nLen= sizeof (struct sockaddr), nRet, s1, s2,
       noLine, link, cnt, i, err;
   long lRet;
   char szBuf[256], szLine[256];
   struct sockaddr_in saServer, saClient;

   // Check for port argument. Print usage if no port number...

   if (argc != 2) {
      printf ("\n  Usage.: msrtsvr2 PortNumber");
      printf ("\n  Sample: msrtsvr2 5000<enter>\n");
      return (-1);
   }

   // Set SC410 for DIL/NetPC PIO port A = output

   ioperm (CSCIR, 2, 1);
   outb (PAMR, CSCIR);   // Set index pointer to index register A5h
   outb (0xff, CSCDR);   // Index register A5h= 0xff
   outb (PADR, CSCIR);   // Set index pointer to index register A9h
   outb (0x00,CSCDR);    // Index register A9h= 0 (port A = 0x00)
   ioperm (CSCIR, 2, 0);

   // Create a TCP/IP stream socket...

   s1= socket (AF_INET, SOCK_STREAM, 0);

   // ... and check for error...

   if (s1 < 0){
      printf ("\n  Can not create socket.\n");
      return (-1);
   }

   // Fill in the server internet address structure...

   saServer.sin_family= AF_INET;               // Address family
   saServer.sin_addr.s_addr= INADDR_ANY;       // Any IP address
   saServer.sin_port= htons (atoi (argv[1]));  // Portnumber
   memset (&(saServer.sin_zero), 0x00, 8);     // Zero the rest...
```

```
// Bind IP address/portnumber to socket...

nRet= bind (s1, (struct sockaddr *) &saServer,
                 sizeof (struct sockaddr));

// ... and check for any errors...

if (nRet < 0) {
   printf ("\n  Can not bind socket to address structure.\n");
   close (s1);
   return (-1);
}

// Execute a listen (queue client connect requests)...

nRet= listen (s1, 5);

// ... and check for any errors...

if (nRet < 0) {
   printf ("\n  Can not use listen function.\n");
   close (s1);
   return (-1);
}

// Server loop. Wait for client request without end...

while (1) {

   // Waiting for connect from client...

   printf ("\n  Waiting for connect...");
   fflush (stdout);
   if ((s2= accept (s1, (struct sockaddr *) &saClient, &nLen)) < 0) {
      printf ("\n  Can not use accept function.\n");
      close (s1);
      close (s2);
      return (-1);
   }
   printf ("\n  Connect to %s\n", inet_ntoa (saClient.sin_addr));
   fflush (stdout);

   // Send usage message to client...

   sprintf (szBuf, "MSR-Telnet-Server. Vers. 1.00 for …");
   send (s2, szBuf, strlen (szBuf), 0);
   sprintf (szBuf, "Enter number 0 - 255 and press enter …");
```

```
send (s2, szBuf, strlen (szBuf), 0);

// We have a link...

link= 1;
while (link) {

   // Get next line with number or CTRL-C...

   noLine= 1;
   err= 0;
   cnt= 0;
   sprintf (szBuf, ">");
   send (s2, szBuf, strlen (szBuf), 0);
   memset (szLine, 0, sizeof (szBuf));
   while (noLine) {

      // Wait for client data...

      memset (szBuf, 0, sizeof (szBuf));
      nRet= recv (s2, szBuf, sizeof (szBuf), 0);

      // Watch the link state...

      if (nRet <= 0) {
         link= 0;                    // Diconnect by client
         noLine=0;
      }
      else {

         // Remove all char´ s from szBuf...

         i= 0;
         while (nRet != 0 && noLine != 0 && link != 0) {

            // If valid char: save data...

            if (szBuf[i] >= 0x30 && szBuf[i] <= 0x39) {
               szLine[cnt]= toupper (szBuf[i]);
               cnt++;
            }
            else {

               // Count invalid char´ s...

               if (szBuf[i] != '\r' && szBuf[i] != 0x04)
                  err++;
```

```
                }

                // Check for line end ('\r') or max char count...

                if (szBuf[i] == '\r' || cnt > 16)
                   noLine= 0;

                // Check for session end (CTRL-D)...

                if (szBuf[i] == 0x04) {
                   link= 0;
                   noLine=0;
                }

                // Next char...

                i++;
                nRet--;
             }
          }
       }

       // We have a new line...

       if (link == 1) {

          // Check for valid number (0 - 255)...

          if (strlen (szLine) == 0 || atoi (szLine) > 255 || err != 0)
             sprintf (szBuf, "ERR\n\r");
          else {
             sprintf (szBuf, "OK\n\r");

             // Output operation: write value to port A

             ioperm (CSCIR, 2, 1);
             outb (PADR, CSCIR);
             outb (atoi (szLine) & 0xff,CSCDR);
             ioperm (CSCIR,2,0);
          }
          send (s2, szBuf, strlen (szBuf), 0);
       }
    }

    // Close data socket...

    close (s2);
```

```
    }
}
```

- **2. Step:** Build a executable form the source code. Use the gcc (GNU C Compiler) direct with a command line. Enter the following command line for building the executable

  ```
  gcc -O -o msrtsvr2 msrtsvr2.c
  ```

- **3. Step:** Transfer the executable for the measurement Telnet server from your development system to the (A)DNP/1486. You can use a FTP session for this file transfer.

- **4. Step:** Run the measurement Telnet server on your (A)DNP/1486. Make sure to have superuser rights. Then enter

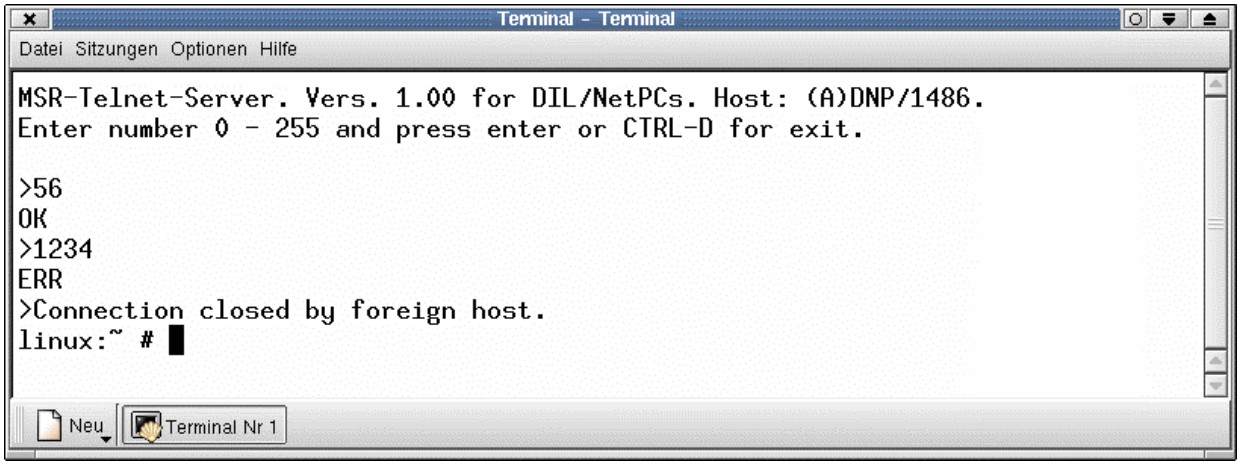
  ```
  ./msrtsvr2 5000
  ```

  "5000" is a TCP port number. You can use any other unused port number, but not "23". This is the standard port number for Telnet servers. Please note, that you have a other Telnet server currently running. This server is using the TCP port number 23. A good choice are port numbers over 1024.

- **5. Step:** Now you can test the measurement Telnet server. Run a new Telnet client program on your development system. Then enter

  ```
  telnet 192.168.0.126 5000
  ```

  within a text console window. After that a Telnet client for TCP port number 5000 starts and builds a TCP connection. Within the new Telnet client window you see the sign-in message of the measurement Telnet server. Now you can enter valid numbers. Watch the LEDs of your DNP/EVA1 or 2 evaluation board.



For disconnect the TCP connection between the Telnet server and the Telnet client, please use the key combination CTRL-D.