Socket Connections for LAN enabled Instruments

Agilent Technologies

Table of Contents

1. Introduction 3

1.1. Assumptions 3

1.2. The Basics 3

2. SCPI Connection 4

3. Control Connections 5

3.1. Device Clear 5

3.2. Service Request 5

4. Example C/C++ Code 6

4.1. Open SCPI Socket Connection 6

4.2. Reading/Writing On SCPI Socket Connection 7

4.3. Open Control Socket Connection 8

4.3.1. Sending Device Clear 9

4.3.2. Service Request 10

1. Introduction

1.1. Assumptions

This document is NOT a tutorial on Sockets or Sockets protocol. It is assumed that the reader is familiar with Sockets and TCP/IP protocols. This document also assumes that the reader is familiar with SCPI and the SCPI status system.

1.2. The Basics

Socket connections for instruments are a convenient way to communicate with LAN enabled instrument without going through any I/O libraries. They are easy to use and are very fast for transporting large blocks of data. Plus sockets are supported on a wide variety of OS's and machines.

2. SCPI Connection

Agilent instruments have standardized on using port 5025 for SCPI socket services. Once a connection is made you simply send the SCPI strings to the instrument and read back responses over the socket connection. All “Program Messages” must be terminated with a newline for the message to be parsed. All responses that are read will also have a newline at the end.

3. Control Connections

Socket connections support a control connection. This second connection is used by client to send Device Clear and to listen for Service Requests. The initial connection is used to send and receive ASCII/SCPI commands, queries, and query responses. The user needs to query the initial connection for its control connection port number. The SCPI command for this query is:

SYSTem:COMMunicate:TCPip:CONTrol?

If this SCPI query fails, it means that the instrument does not support the Socket Control connection. Once the user has this port number they simply open another socket connection to the instrument.

3.1. Device Clear

To use the control connection to do a device clear, send the string “DCL\n” to the device. When the device clear has been completed the instrument will send the string “DCL\n” back to the client. The ***handshake*** is needed so that the client knows when the device clear has completed.

3.2. Service Request

The user would enable service requests exactly the same as for GPIB. Once service requests have been enabled just listen on the control connection. When service requests have been encountered the instrument will send the string “SRQ +nn\n” to the client. The nn is the status byte for the client to use to determine the source of the service request.

4. Example C/C++ Code

This example code was written on Windows OS using the WinSock library. The WinSock library is mostly compliant with Berkeley sockets and prefixes non-compliant calls with “WSA”.

4.1. Open SCPI Socket Connection

The open socket connection is using the default 5025 port number and a bogus IP address for the hostname. Except for the WSAStartup the rest of the socket commands are Berkeley compliant.

```
void OpenInstrument(SOCKET & Socket)
{
        int portno = 5025;
        int pHostname = “156.140.90.163”;
        struct hostent *hp;
        struct sockaddr_in sockaddr;
        int sendsockbufsize=52224;
        int recvsockbufsize=52224;
        int on=1;
        WORD wVersionRequested;
        WSADATA wsaData;

        // Set to bad value
        Socket = 0;

        // Initialize WinSock
        wVersionRequested = MAKEWORD(1, 1);
        int err = WSAStartup(wVersionRequested, &wsaData);
        if (err != 0)
        {
           printf("Could not initialize WinSock\n");
           return;
        }

        // Determine the IP address we are connecting
        hp = gethostbyname(pHostname);
        if (hp == NULL)
        {
           printf("Could not resolve hostname %s into IP address.\n", pHostname);
           return;
        }

        sockaddr.sin_addr.s_addr = ((struct in_addr *)(hp->h_addr))->s_addr;

        // Create a socket endpoint
        if ( (Socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
           printf("Can't create socket\n");
           return;
        }
```

```
        // Fill in the socket address structure
        sockaddr.sin_family = AF_INET;
        sockaddr.sin_port = htons(portno);

        // Adjust the recv and send buffer size of the socket
        // Ignore errors because the op sys might not support what we ask for
        setsockopt(Socket, SOL_SOCKET, SO_SNDBUF,
                (char *)&sendsockbufsize, sizeof(int));
        setsockopt(Socket, SOL_SOCKET, SO_RCVBUF,
                (char *)&recvsockbufsize, sizeof(int));

        // Turn on TCP No Delay option
        setsockopt(Socket, IPPROTO_TCP, TCP_NODELAY, (char *)&on, sizeof(on));

        // Connect to server
        if (connect(Socket,(const struct sockaddr *)&sockaddr,sizeof(sockaddr)) < 0)
        {
           printf("Could not connect to %s on port %d\n", czHostname, portno);
           return;
        }

        // Print that we are running
        printf("Running over TCP/IP to %s port %d\n", hp->h_name, portno);
}
```

4.2. Reading/Writing SCPI Socket Connection

```
int SocketRead(SOCKET Socket, char *buffer, long length)
{
   int result;

   // Read the response now (either the entire thing or partial)
   if ((result = recv(Socket, (char *)buffer, length, 0)) <= 0)
   {
      printf("Read error\n");
      return 0;
   }
   else
   {
      buffer[result] = '\0';  // terminate buffer!
   }

   return result;
}

int SocketWrite(SOCKET Socket, char *buffer)
{
   int result;
   long length = strlen(buffer);

   // Write the data now
   if ((result = send(Socket, (const char *)buffer, length, 0)) <= 0)
   {
```

```
        printf("Write error");
        return 0;
    }

    return result;
}
```

4.3. Open Control Socket Connection

Opening the control socket requires getting the control socket port number from the current SCPI socket. Also, the hostname is the same as the current socket. After opening the Socket a newline must be sent and a newline received to initialize the connection.

```
void OpenControl(SOCKET Socket, char *pHostname, SOCKET &SocketControl)
{
    char portBuf[100];

    // Ask this socket what its control port number is?
    SocketWrite(Socket, "SYST:COMM:TCPIP:CONTROL?\n");
    SocketRead(Socket, portBuf, 32);
    controlPortNo = atoi(portBuf);

    if (controlPortNo == 0)
    {
        printf("Host did not return valid portNo %d.\n", controlPortNo);
    }
    else
    {
        struct sockaddr_in sockaddr;
        struct hostent *hp;

        // Print the control channel port number
        printf("Control port = %d\n", controlPortNo);

        // Determine the IP address we are connecting
        hp = gethostbyname(pHostname);
        if (hp == NULL)
        {
            printf("Could not resolve hostname %s into IP address.\n", pHostname);
            return;
        }

        sockaddr.sin_addr.s_addr = ((struct in_addr *)(hp->h_addr))->s_addr;

        // Create a socket endpoint
        if ( (SocketControl = socket(AF_INET,SOCK_STREAM,0)) < 0)
        {
            printf("Could not create a socket\n");
            return;
        }

        /*  */
        sockaddr.sin_family=AF_INET;
        sockaddr.sin_port=htons((unsigned short)controlPortNo);
```

```
        // Setup a control connection with the instrument
        if (connect(SocketControl,(const struct sockaddr *)&sockaddr,
                        sizeof(sockaddr))<0) {
            printf("Could not establish Control connection to port %d on %s\n",
                    ControlPortNo, czHostname);
            return;
        }

        // Send a new-line on the control connection
        // to make sure the control connection is alive
        if (send(SocketControl, "\n", 1, 0) != 1)
        {
            printf("write on control port failed!\n");
            return;
        }

        // Read back a new-line indicating that the
        // control channel is alive
        if (recv(SocketControl, (char *)portBuf, 100, 0) <= 0)
        {
            printf("read on control port failed!\n");
            return;
        }

        // Print that we are running
        printf("Running control TCP/IP to %s port %d\n", pHostname, controlPortNo);
    }

    return;
}
```

4.3.1. Sending Device Clear

The device clear requires the client to send a “DCL\n” string to the instrument. When the instrument has completed the device clear it will return the string “DCL\n”.

```
void DeviceClear(SOCKET SocketControl)
{
    char resp[12];

    ControlWrite("DCL\n");
    //
    // Instrument will return “DCL\n” when complete.
    ControlRead(SocketControl, resp, 12);
    printf("DeviceClear resp=%s\n", resp);
}
```

4.3.2. Service Request

```
#define ALARM 2
#define ERR   4
#define QDATA 8
#define MAV   16
#define STDEV 32
#define STDOP 128

DWORD WINAPI ReadSRQs(SOCKET SocketControl)
{
   char srqBuf[64];
   int srqValue;

   if (SocketRead(SocketControl, srqBuf, 64) == 0)
      returnf  -1;

   printf("Returned %s\n", srqBuf);
   //
   // Just in case.
   if (strncmp(srqBuf, "SRQ", 3) == 0)
   {
      sscanf(srqBuf, "SRQ %d\n", &srqValue);

      if (ALARM & srqValue)
         printf("Alarm ");
      if (ERR & srqValue)
         printf("Error ");
      if (QDATA & srqValue)
         printf ("Quest Data ");
      if (MAV & srqValue)
         printf("Message Avalable ");
      if (STDEV & srqValue)
         printf("Standard Event ");
      if (STDOP & srqValue)
         printf("Standard Operation");
      printf("\n");
   }

   return 0;
}
```