

Asynchronous Socket Server data

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2006-04/msg00696.html>

- *From:* Macca <Macca@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 5 Apr 2006 07:58:02 -0700
-

Hi,

I am writing an asynchronous socket server to handle 20+ simultaneous connections.

I have used the example in MSDN as a base. The code is shown at end of question.

Each connection has a number of different types of data coming in. I have a databuffer for each type of data coming in.

The socket server knows what type of data it expects due to the interface protocol I have implemented.

The main part of this protocol is that the client sends a 5 byte message which indicates the type of data to follow.

The client then sends the data in response to the server acking the 5 byte message.

This data will be split up into packets. Each packet is checked and a valid reply from the server will prompt the client to send the next packet until the whole data has been received.

The socket server knows what type of data to expect and will therefore put it into the appropriate sized data buffer.

My question is, rather than have 20 copies of the data buffer for datatype1, 20 copies for datatype2 etc is there anyway

I can have 1 databuffer only for each datatype to handle multiple connections?

As each instance of my app could have a different number of connections to it, the above would stop the need for adding or removing buffers for each instance of my app if the number of different connections differs from instance to instance.

I'd appreciate any suggestions,

Thanks In Advance
Macca

Asynchronous Socket Server data

```
public sealed class AsynchronousSocketServer
{

// ReadCallback variables. Will declaring these variables here
enables multiple connections
// to use them?

static byte[] messagePacketArray = new
byte[SysConstants.MessagePacketSize];
static byte[] standbyMessageArray = new
byte[SysConstants.StandbyMessageSize];
static byte[] roiPacketArray = new byte[SysConstants.RoiPacketSize];
static byte[] diagnosticPacketArray = new
byte[SysConstants.DiagnosticPacketSize];
static byte[] roomStatusArray = new
byte[SysConstants.roomStatusPacketSize];
static byte[] readyToReceiveArray = new
byte[SysConstants.readyToReceiveSetupPacketSize];

static int packetIndex;

// Sensor Ack/Nack buffers
static byte[] ackArray = new byte[1] { SysConstants.Ack };
static byte[] nackArray = new byte[1] { SysConstants.Nack };
// Reply Message Buffers
static byte[] standbyArray = new byte[3] { SysConstants.Standby, 0,
0 };

static ushort diagnosticPacketCounter;

// Thread signal.
private static ST.ManualResetEvent allDone = new
ST.ManualResetEvent(false);

/// <summary>
/// Default Constructor
/// </summary>
private AsynchronousSocketServer()
{
}

/// <summary>
/// Listen for incoming connections from the Sensor Clients
/// </summary>
public static void StartListening()
{
// Establish the local endpoint for the socket.

string hostname = "";
```

Asynchronous Socket Server data

```
SN.IPHostEntry ipHostInfo = SN.Dns.GetHostEntry(hostname);
SN.IPAddress ipAddress = ipHostInfo.AddressList[0];
SN.IPEndPoint localEndPoint = new SN.IPEndPoint(ipAddress,
SysConstants.TcpPortNumber);

// Create a TCP/IP socket.
SNS.Socket listener = new
SNS.Socket(SNS.AddressFamily.InterNetwork,
SNS.SocketType.Stream, SNS.ProtocolType.Tcp);

// Bind the socket to the local endpoint and listen for incoming
connections.
try
{
listener.Bind(localEndPoint);
listener.Listen(SysConstants.ClientConnectionsMaxBufferSize);

while (true)
{
// Set the event to nonsignaled state.
allDone.Reset();

// Start an asynchronous socket to listen for connections.
listener.BeginAccept(
new AsyncCallback(AcceptCallback),
listener);

// Wait until a connection is made before continuing.
allDone.WaitOne();
}

}
catch (SNS.SocketException e)
{
Console.WriteLine(e.ToString());
}

}

/// <summary>
/// Callback Method that handles client connections
/// </summary>
/// <param name="ar"> The Interface that contains Socket
Information</param>
public static void AcceptCallback(IAsyncResult ar)
{
if (ar == null)
{
throw new ArgumentNullException("ar");
}
}
```

Asynchronous Socket Server data

```
// Signal the main thread to continue.
allDone.Set();

// Get the socket that handles the client request.
SNS.Socket listener = (SNS.Socket)ar.AsyncState;
SNS.Socket handler = listener.EndAccept(ar);

// Create the state object.
StateObject state = new StateObject();
state.workSocket = handler;
state.dataType = SysConstants.WaitingForCommand;
state.dataSize = SysConstants.MessagePacketSize;

handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
new AsyncCallback(ReadCallback), state);
}

/// <summary>
/// Callback Method that processes data sent over socket connection
/// by the sensor.
/// </summary>
/// <param name="ar"></param>
public static void ReadCallback(IAsyncResult ar)
{
    if (ar == null)
    {
        throw new ArgumentNullException("ar");
    }

    // Determine which sensor has sent data

    // Retrieve the state object and the handler socket
    // from the asynchronous state object.
    StateObject state = (StateObject)ar.AsyncState;
    SNS.Socket handler = state.workSocket;

    // Read data from the client socket.
    int bytesRead = handler.EndReceive(ar);

    if (bytesRead > 0)
    {
        try
        {
            // Check what type of data has been received
            switch (state.dataType)
            {
                // Build message if total bytes expected not reached yet
                case SysConstants.Message:
                    Array.Copy(state.buffer, 0, messagePacketArray,
                    packetIndex, bytesRead);
```

Asynchronous Socket Server data

```
break;

case SysConstants.WaitingForCommand:
Array.Copy(state.buffer, 0, standbyMessageArray,
packetIndex, bytesRead);
break;

case SysConstants.RoiProcessed:
Array.Copy(state.buffer, 0, roiPacketArray,
packetIndex, bytesRead);
break;

case SysConstants.DiagnosticImage:
Array.Copy(state.buffer, 0,
diagnosticPacketArray, packetIndex, bytesRead);
break;

case SysConstants.RoomStatusData:
Array.Copy(state.buffer, 0, roomStatusArray,
packetIndex, bytesRead);
break;

case SysConstants.ReadyToReceiveSetupData:
Array.Copy(state.buffer, 0, readyToReceiveArray,
packetIndex, bytesRead);
break;

}
}
catch (System.ArgumentException e)
{
Console.WriteLine("{0} : error ", e.Message);
}

state.totalBytesRead += bytesRead; // Used to indicate when
full packet is received
packetIndex += bytesRead;

if (state.totalBytesRead == state.dataSize)
{
// Full Packet has been received
packetIndex = 0;
state.totalBytesRead = 0;
switch (state.dataType)
{
case SysConstants.WaitingForCommand:
//
// Do Something
```

Asynchronous Socket Server data

```
//  
break;  
  
case SysConstants.Message:  
//  
// Do Something  
//  
break;  
  
case SysConstants.RoiProcessed:  
//  
// Do Something  
//  
break;  
  
case SysConstants.DiagnosticImage:  
//  
// Do something  
//  
break;  
  
case SysConstants.RoomStatusData:  
//  
// Do Something  
//  
break;  
  
case SysConstants.ReadyToReceiveSetupData:  
//  
// Do Something  
//  
break;  
}  
// Check Socket for more data  
handler.BeginReceive(state.buffer, 0,  
StateObject.BufferSize, 0,  
new AsyncCallback(ReadCallback), state);  
  
}  
else  
{  
// Full packet has not been received yet  
handler.BeginReceive(state.buffer, 0,  
StateObject.BufferSize, 0,  
new AsyncCallback(ReadCallback), state);  
}  
}  
else  
{  
}  
}
```

Asynchronous Socket Server data

```
private static void Send(SNS.Socket handler, byte[] byteData)
{
    // Generate the CRC for data
    Int16 crc = CrcGenerator.GenerateCrc(byteData, byteData.Length);

    // Add CRC to end Message Packet
    byteData[byteData.Length-1] = (byte)(crc >> 8);
    byteData[byteData.Length-2] = (byte)crc;

    // Begin sending the data to the remote device.
    handler.BeginSend(byteData, 0, byteData.Length, 0,
    //handler.BeginSend(byteData, 0, 1, 0,
    new AsyncCallback(SendCallback), handler);
}

private static void SendCallback(IAsyncResult ar)
{
    try
    {
        // Retrieve the socket from the state object.
        SNS.Socket handler = (SNS.Socket)ar.AsyncState;

        // Complete sending the data to the remote device.
        int bytesSent = handler.EndSend(ar);
        Console.WriteLine("Sent {0} bytes to client.", bytesSent);

        //handler.Shutdown(SNS.SocketShutdown.Both);
        //handler.Close();

    }
    catch (System.ArgumentException e)
    {
        Console.WriteLine(e.ToString());
    }
    catch (SNS.SocketException e)
    {
        Console.WriteLine(e.ToString());
    }
}

/// <summary>
/// State object for reading client data asynchronously
/// </summary>
public class StateObject
{
    // Client socket.
    internal SNS.Socket workSocket;
```

Asynchronous Socket Server data

```
// Size of receive buffer.  
public const int BufferSize = 1024;  
// Receive buffer.  
internal byte[] buffer = new byte[BufferSize];  
// identify if message or data packet  
//internal bool messagePacket;  
internal byte dataType;  
  
// Total Bytes read  
internal Int32 totalBytesRead;  
// Size of Data Packet expected  
internal Int32 dataSize;  
}  
.
```