

Communications Breakdown with CSocket

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.cpp/2004-06/1490.html

From: Brendan Grant (grantb_at_dahat.com)

Date: 06/10/04

Date: 10 Jun 2004 10:46:07 -0700

The following describes a problem I have been having for the better part of 3 months, for approximately the last month however I have been focused solely on solving it and seem to be no nearer now than a month ago. Any thoughts or suggestions that might lead to a fix would be greatly appreciated.

I have to VB based applications on two different PC's, a client and a server which use a pair of OCX's to provide a communication method between the two and which use an overloaded version of the CSocket class.

A problem has arisen where communication between the two OCX's will be interrupted. The socket will appear to remain open as the Disconnection event on either side has not been fired, however neither side is able to hear the other (even when sending on the other side is occurring).

While not 100% predictable, the only times the communication breaks down is when the method SendPlayList or one of a couple other Server side OCX methods fire which send a string. Note: These functions do not cause a break down each time they are called, but seem to ultimately be responsible for the break down.

In trying to track down this issue, I have implemented critical sections on both the client and server side (sending and receiving both) to help to ensure that multiple sends or processing of received data does not occur. I have also implemented basic checking (to be removed later) to ensure that the correct amount of data is being sent and received each time.

Below is some of the code being worked with:

On the VB server side, we have the server OCX instantiated and named Remote, we call the SendPlayList method of it, passing the string sPlayList as such:

```
Remote.SendPlayList sPlayList
```

Within the server side OCX, the above line calls the following function:

```
void CSRemoteCtrl::SendPlayList(LPCTSTR PlayList)
{
    PACKET_HEADER hdr;
    BYTE* pData;
    int iBufSize;
    SendCriticalSection.Lock();
    if (lstrlen(PlayList) > 0)
    {
        if (m_pClientSocket != NULL)
        {
            hdr.cmd = SERVER_PLAY_LIST;
            hdr.size = lstrlen(PlayList)+1; // +1 for NULL terminator

            iBufSize = sizeof(PACKET_HEADER) + hdr.size;

            pData = new BYTE[iBufSize];
            memcpy( &pData[0], &hdr, sizeof( PACKET_HEADER ));
            memcpy( &pData[sizeof(PACKET_HEADER)], PlayList, hdr.size);

            int iRet = m_pClientSocket->Send(pData, iBufSize);
            if(iRet != iBufSize)
                AfxMessageBox("Disconnected due to wrong count of bytes sent");

            delete [] pData;
        }
        else
        {
            hdr.cmd = SERVER_PLAY_LIST;
            hdr.size = 0;

            int iRet = m_pClientSocket->Send(&hdr, sizeof(PACKET_HEADER));
            if(iRet != sizeof(PACKET_HEADER))
                AfxMessageBox("Disconnected due to wrong count of bytes sent");
        }
    }
}
```

On the receiving side within the client OCX, the OnReceive function fires when new data arrives, inside of it there is a switch statement which contains the following:

```
Receive( &packetCommand, sizeof( packetCommand ), MSG_PEEK );

switch( packetCommand )
{

//... Other Case Statements
```

```

case SERVER_PLAY_LIST:
{
    PACKET_HEADER packet;

    // Receive the packet structure
    iRet = Receive( &packet, sizeof( packet ));
if(iRet != sizeof(packet))
    AfxMessageBox("Size Error 1 in: SERVER_PLAY_LIST");
    // Catch the file if it is attached
    if( packet.size > 0)
    {
        char *filenames = new char[ packet.size ];
        iRet = Receive( filenames, packet.size );
if((UINT32)iRet != packet.size)
    AfxMessageBox("Size Error 2 in: SERVER_PLAY_LIST");
        // Activate the event
        pCRemoteCtrl->FirePLAYLIST( handle, filenames );

        delete filenames;
    }
    else
    {
        // Activate the event
        pCRemoteCtrl->FirePLAYLIST( handle, "" );
    }
}
break;
//... Other Case Statements

}

```

A couple of conventions being used here is that the packets I am using have a header sections which contain an enumerated value which describes the kind of packet it is, based on that, a different type of payload packet will be used to read the data and process it, similar to how the sending side works.

Most of the time, the above code, along with similar blocks functions flawlessly, but now and then, this or several others can and does bring down the communication.

Also, this code is the 3rd version of code which has been used in the same way (providing TCP/IP connectivity between VB apps) and is the first to exhibit these problems, thus I require an easy to implement solution which does not require rewriting or modifying large sections of it.