LISSimV5 Overview

High-level Overview

Languages: C++, C++/CLI, C#

Purpose: To create a server program (LIS) to facilitate and test simultaneous communication with multiple clients (PANTHER instruments) for Hologic, to be integrated with the existing C# GUI in the LIS Simulator. The messaging protocol supported by this project is specified in the Consumer Technical Bulletin and the documentation for Specification of Message Transfer between Clinical Lab Instruments and Computer Systems.

Features: Multithreaded server and client application using Winsock/socket programming and STL threads, C++/CLI wrapper classes to transition from managed to unmanaged code and vice versa, XML documentation for methods used in each class

Project Structure

1. **HostQueryV5 (AssayV5, HeaderV5, PatientInformationV5)**: AssayV5 serves as an example for processing assay requests, HeaderV5 provides the infrastructure for a header response from the server, and PatientInformationV5 contains relevant information for each patient corresponding to a sample ID.
2. **TCPConnectionV5 (TCPConnectionV5):** A TCPConnectionV5 base class contains overlapping operations used by a client and server program for TCP/IP socket programming, built as a static library to reference in TCPServerV5 and TCPClientV5.
3. **TCPClientV5 (TCPClientV5)**: Project simulating the behavior of the PANTHER instruments by initializing a client socket to connect with a server program (LIS).
4. **TCPServerV5 (TCPServerV5)**: Project simulating LIS behavior by initializing a server socket to connect with clients (PANTHER instruments).
5. **TCPcsLibrary (csLib)**: Project that contains a C# class with functions, built as a C# DLL, to be called from TCPUnmanagedWrapper/TCPUnmanagedWrapperTest, testing functionality.
6. **TCPUnmanagedWrapper (UnmanagedWrapper)**: Project that contains a wrapper class to export a C# class (managed code) to be called from an unmanaged C++ program.
7. **TCPUnmanagedWrapperTest (UnmanagedWrapperRunner)**: Project that tests the ability to call TCPcsLibrary's class's functions from an unmanaged C++ application, using TCPUnmanagedWrapper as an intermediary.
8. **UnitTests (CheckSumTests)**: Project that contains the verifications that CheckSum function works; also used to generate checksums for header/patient/assay messages.
9. **TCPWrapperClassV5 (ManagedObject, TCPWrapperClassV5)**: C++/CLI Project acting as an intermediary between TCPConnectionV5 and TCPV5CSharpApp, calling TCPConnectionV5 (an unmanaged C++ static library) functions from a C# application.

10. **TCPV5CSharpApp (Program)**: C# project/application to test the ability to instantiate a TCPConnectionV5 object through the wrapper class in managed code (C#).

Key Classes & Functionality

1. *TCPConnectionV5*: functions roughly divided into initialization/shutdown server/client, utility methods, send/receive messages, testing broadcast protocol, testing host-query exchange, and
    a. **Key Variables**: Patient host-query exchange database (map), host-query exchange Sample ID's, host-query exchange assay vector database (map), corresponding patient/assay vector databases & Sample ID's for broadcast, socket/messaging variables, default values for headers/delimiters & contention defaults
    b. **Initialization/shutdown**: initWinsock, findClientMessager, startClient, startServer
    c. **Utility methods**: printTime, printMessage, getTime, setter/accessor methods for variables, generateChecksum
    d. **Send/receive messages**: sendMessage, sendEOT, sendENQ, sendACK, receiveMessage
    e. **Test broadcast protocol**: makeBroadcastDatabase, sendBroadcast, receiveBroadcast
    f. **Test host-query exchange**: senderExchange, receiveExchange, serverIdle, exchange
2. *TCPWrapperClassV5*: A subclass of ManagedObject facilitating calling TCPConnectionV5 methods (unmanaged C++) from a C# application. Functions roughly divide into the following: start/end client connection, set header message, server patient/assay database manipulation, utility methods
    a. **Start/End Client Connection**: listen, close
    b. **Set Header Message**: setHeaderFrom, setHeaderTo, setHeaders, setDelimiters
    c. **Patient/Assay Database Manipulation**: setMaximumDatabaseSize, addSampleOrder, deleteSampleOrder
    d. **Utility**: setLineBids, setAutoOrder, printTime, printMessage

Dependencies/Relations

1. **TCPConnectionV5**: Library uses HostQueryV5 to create a patient/assay vector database and generate messages corresponding to patients/assays
    a. References: HostQueryV5
2. **TCPServerV5**: TCPServerV5 creates an instance of the TCPConnectionV5 class, using the server startup/messaging protocols specified. Instance of TCPUnmanagedWrapper is instantiated used to prove viability of wrapper class use of C# DLL class methods.

    a. Linker: includes TCPUnmanagedWrapper.lib under input's additional dependencies, and the additional library directory under general properties configured to folder which contains TCPUnmanagedWrapper
    b. References: HostQueryV5, TCPConnectionV5, TCPUnmanagedWrapper.
3. **TCPClientV5**: TCPClientV5 creates an instance of the TCPConnectionV5 class (specifying client startup/messaging protocol)
    a. References: HostQueryV5 and TCPConnectionV5
4. **TCPWrapperClassV5**: Uses the methods of TCPConnectionV5, which pulls from HostQueryV5. Creates a managed instance of TCPConnectionV5
    a. References: HostQueryV5 and TCPConnectionV5
5. **TCPV5CSharpApp**: Creates an instance of TCPWrapperClassV5 to call TCPConnectionV5 methods.
    a. References: TCPWrapperClassV5
6. **TCPUnmanagedWrapper**: Creates a CLI/C++ wrapper instance of csLib in the TCPcsLibrary project (a C# DLL)
    a. References: TCPcsLibrary
7. **TCPUnmanagedWrapperTest**: Creates an instance of TCPUnmanagedWrapper in managed C# code. Requires additional configuration for linker.
    a. Linker: same as TCPServerV5
    b. References: TCPUnmanagedWrapper
8. **UnitTests**: Generates the checksum quantities that are used in TCPConnectionV5 for assay/patient messages.
    a. References: TCPConnectionV5

<u>Testing</u>

Self-Tests: Testing server/client applications on one computer

- *Step 1*: Configure the IP Address/desired port number for the client/server
- *Step 2*: Choose whether to test broadcast (create a patient/assay vector database of size 50k, corresponding to unique Sample ID's, then time + test with any number of clients) OR test exchange (engaging in host-query exchange--receiving a query, issuing a response, and then the server idling)
    - Broadcast test focuses on timing (ensure that databases of up to 50k patients and assay vectors are supported & timing remains consistent for 1-16 clients), host-query exchange test focuses on the server program sending the correct messages in response to initial queries
- *Step 3*: If testing 1 client for a server, F5. Default startup configuration (multiple startup projects) enables TCPServerV5 and TCPClientV5. The server must be live before a client

can connect. To configure a different number of clients for broadcast or host-query exchange, edit and run Multiclient test.bat in the Debug folder.

Testing with the Panther PC

- *Step 1*: Configure the IP Address/port # for the server application and the Panther PC
- *Step 2*: Initialize a connection from the Panther PC (ensure server is up and running)
- *Step 3*: Send a query, configured to a Sample ID from the Panther PC and ensure that the appropriate patient/assay is delivered.
- *Step 4*: Test the Panther PC against the existing LIS Simulator to ensure that the same messaging verdicts are reached (for both positive & negative tests--when a patient/assay exists for a given sample ID and when one does not)

Current Limitations/Potential Improvements

1. **List to Vector conversion**: Currently, addSampleOrder and setAutoOrder take in vector<AssayV5> instances for their methods, when in reality, the C# application would pass a List<AssayV5> instance. One potential workaround would be to pass the string entries to create patient/assay vector instances, corresponding to a sample ID.
2. **Broadcast timing**: Broadcast method is intended to support up to 16 clients (using multithreading--each client on their own, independent thread) without significant time disparities between supporting 1 client and supporting 16. While the differences ranging from 1 to 5 clients are negligible, the difference between 1 client and 16 clients is noticeable (magnitude of greater than sixteen-fold).
3. **Expanding UnmanagedWrapper Functionality**: Currently, TCPServerV5 and UnmanagedWrapperTester call basic print/string manipulation functions from csLib (C# DLL class functions) through the CLI/C++ wrapper. Work to call ClientConnected function and ClientClosed function to indicate when a client has formed a new connection or closed connection.
4. **Message Logging**: Currently, <CR> character should be displayed as the character and as a new line. Creates a potential source of discrepancy when printing a message vs. generating the log, depending on the type of message sent/received.
5. **Pipes**: Integrating the C# and C++ named pipes projects to facilitate the transfer of the message logs.