Cameron's networking library(TCP)

Note: This library uses the library zlib, and I am not, nor do I claim to the writer of it.

Network:

CAMSNETLIB int InitializeNetworking();

Initializes networking.

Remarks:
- Must be called before you can do anything(lol)

Return value:
- Zero:
    o When function succeeds.
- Non Zero:
    o See WSAStartup.

CAMSNETLIB int CleanupNetworking();

Cleans up networking.

Return value:
- Zero:
    o When function succeeds.
- Non Zero:
    o See WSACleanup.
Remarks:
- Must be called for every InitializeNetworking call.

Client:

CAMSNETLIB TCPClientInterface* CreateClient(cfunc msgHandler, dcfunc disconFunc, int compression = 9, float pingInterval = 30.0f, void* obj = nullptr);
Parameters:
- msgHandler – Pointer to a function with the following signature:
    o void MsgHandler(TCPClientInterface& clint, const BYTE* data, DWORD nBytes, void* obj)
    o This is where all packets are received.
- disconFunc – Pointer to a function with the following signature:
    o void function(bool unexpected)

- o Function called when you get disconnected from server.
- [optional] compression – This sets level what compression the client will compress data to send at. Value of 1-9.
- [optional] pingInterval – Interval at which client pings server, used to detect half dropped connections(Ping is actually a TCP keep alive message)
- [optional] Obj - Pointer to a class object, that is passed to the msghandler function; it is mostly used in oop.

Creates, and initializes a client object.

Remarks:
- Must be called before you can do anything(lol)
- A call to DestroyClient is required.

Return value:
- TCPClientInterface*

CAMSNETLIB void DestroyClient(TCPClientInterface*& client);

Destroys the specified client object.

virtual bool Connect(const LIB_TCHAR* dest, const LIB_TCHAR* port, bool ipv6 = false, float timeOut = 5.0f) = 0;
Parameters:
- dest - IP address or hostname of server.
- port – Port server is listening on.
- Ipv6 – Specifies if address pointed to by dest is an ipv6 address.
- timeOut – period of time in which function will timeout if a successful connection has not been made.

Attempts to connect to the destination(IP address or hostname).
It waits/blocks until either:
1) A successful connection has been established.
2) Timeout period has returned

Return value:
- True:
    - o When function succeeds.
- False:
    - o If client is already connected, or function fails.

virtual void Shutdown() = 0;

Immediately shuts down connection to server, and performs cleanup.
It waits/blocks until function returns.


```
virtual void Disconnect() = 0;
```

Shuts down connection to server, and performs cleanup.
It does not wait/block.


```
virtual bool RecvServData() = 0;
```

Initializes socket, and receiving thread and starts receiving data from server.

Remarks:
  • Must be called before you can send any type of data to server.

Return value:
  • True:
      o When function succeeds.
  • False:
      o If client is not connected.
      o Receive thread fails to create.


```
virtual void SendServData(const char* data, DWORD nBytes, CompressionType
compType = BESTFIT) = 0;
```

Parameters:
  • data – pointer to data to send.
  • nBytes – byte count of data to be sent
  • compType – compression preference,(BESTFIT[recommended],
    SETCOMPRESSION, NOCOMPRESSION)

Sends data to connected server.


```
virtual HANDLE SendServDataThread(const char* data, DWORD nBytes,
CompressionType compType = BESTFIT) = 0;
```

Parameters:
  • data – pointer to data to send.
  • nBytes – byte count of data to be sent
  • compType – compression preference,(BESTFIT[recommended],
    SETCOMPRESSION, NOCOMPRESSION)

Creates a thread that sends the data specified to the server.

Remarks:
- Handle must be closed after this is called either with WaitAndCloseHandle, or CloseHandle.

Return value:
- A handle to created send thread.

```cpp
virtual void SendMsg(char type, char message) = 0;
virtual void SendMsg(const std::tstring& user, char type, char message) = 0;
```

Sends a message to connected server in the format of TYPE, MESSAGE.
These functions are wrappers to SendServData.
Data is not compressed when calling these functions.

```cpp
virtual void Ping() = 0;
```

Pings the server, should be called in message handler.

```cpp
virtual void SetPingInterval(float interval) = 0;
```

Sets the client's ping interval.

```cpp
virtual float GetPingInterval() const = 0;
```

Return Value:
- Current value of ping interval.

```cpp
virtual void SetFunction(cfunc function) = 0;
```

Sets the clients function/message handler, it is called whenever a message is received.

```cpp
virtual bool IsConnected() const = 0;
```

Return value:
- True:
    - If connected.
- False:

       o If not connected.

virtual Socket& GetHost() = 0;

Returns a reference to the connected socket.

Return value:
- Socket&

virtual void* GetObj() const = 0;

Returns a pointer to the object you specified in constructor.

Return value:
- Void*

Server:

CAMSNETLIB TCPServInterface* CreateServer(sfunc msgHandler, customFunc
conFunc, customFunc disFunc, USHORT maxCon = 20, int compression = 9, float
pingInterval = 30.0f, void* obj = nullptr);
Parameters:
- msgHandler - Pointer to a function with the following signature:
  - void MsgHandler(TCPServInterface& serv, ClientData* const clint,
    const BYTE* data, DWORD nBytes, void* obj)
  - This is where all packets are received.
- conFunc - Pointer to a function with the following signature:
  - void function(ClientData* data)
  - Called after client is added to server.
- disFunc - Pointer to a function with the following signature:
  - void function(ClientData* data)
  - Called before client is removed from server.
- [optional] maxCon – The maximum amount of clients the server can
  support before it sends (TYPE_CHANGE, MSG_CHANGE_SERVERFULL) to the
  connecting client.
- [optional] compression –The level of compression the server will
  compress data to send at. Value of 1-9.
- [optional] pingInterval – The frequency the server sends ping messages
  to connected clients, to keep them from timing out. (Ping is actually a
  TCP keep alive message)
- [optional] Obj - Pointer to a class object, that is passed to the
  msghandler function; it is mostly used in oop.

Creates, and initializes a server object.

Remarks:
- Must be called before you can do anything(lol)
- A call to DestroyServer is required.

Return value:
- TCPServInterface*

CAMSNETLIB void DestroyServer(TCPServInterface*& server);

Destroys the specified server object.

virtual IPv AllowConnections(const LIB_TCHAR* port, IPv ipv = ipboth) = 0;

IPv specifies what ipv(s) to bind host sockets with.
Binds host socket, and creates a thread that waits for connections to the server.

Remarks:
- Must be called before you can send any type of data to server.

Return value IPv:
Returns any combination of any of the following:
- Ipv4:
    o If ipv4 was successfully binded.
- Ipv6:
    o If ipv6 was successfully binded.

- Ipnone:
    o If function has already been called.
    o If nothing was binded successfully.
    o Function fails.

virtual void SendClientData(const char* data, DWORD nBytes, Socket addr, bool single, CompressionType compType = BESTFIT) = 0;
virtual void SendClientData(const char* data, DWORD nBytes, Socket* pcs, USHORT nPcs, CompressionType compType = BESTFIT) = 0;
virtual void SendClientData(const char* data, DWORD nBytes, std::vector<Socket>& pcs, CompressionType compType = BESTFIT) = 0;

virtual HANDLE SendClientDataThread(const char* data, DWORD nBytes, Socket addr, bool single, CompressionType compType = BESTFIT) = 0;

```
virtual HANDLE SendClientDataThread(const char* data, DWORD nBytes, Socket*
pcs, USHORT nPcs, CompressionType compType = BESTFIT) = 0;
virtual HANDLE SendClientDataThread(const char* data, DWORD nBytes,
std::vector<Socket>& pcs, CompressionType compType = BESTFIT) = 0;
```

Parameters:
- data – pointer to data to send.
- nBytes – byte count of data to be sent
- compType – compression preference,(BESTFIT[recommended],
  SETCOMPRESSION, NOCOMPRESSION)

Sends data to specified clients. Threaded functions do this on a separate
thread.

Remarks:
- Handle must be closed after this is called either with
  WaitAndCloseHandle, or CloseHandle.
- First overload, the value of single determines what the function does
  - o If single is true it sends only to address specified.
  - o If single is false, and addr is not connected, it sends to all
    clients currently connected to the server.
  - o If single is false, and addr is connected, it sends to all
    clients, excluding the addr specified.

Return value:
- For threaded functions returns a HANDLE to created thread, non-threaded
  return nothing.


```
virtual void SendMsg(Socket pc, bool single, char type, char message) = 0;
virtual void SendMsg(Socket* pcs, USHORT nPcs, char type, char message) = 0;
virtual void SendMsg(std::vector<Socket>& pcs, char type, char message) = 0;
virtual void SendMsg(const std::tstring& user, char type, char message) = 0;
```

Sends a message to specified clients in the format of TYPE, MESSAGE.
These functions are wrappers to SendClientData.
Data is not compressed when calling these functions.


```
virtual ClientData* FindClient(const std::tstring& user) const = 0;
```

Return value:
- A pointer to the ClientData, specified by user.


```
virtual void DisconnectClient(ClientData* client) = 0;
```

Disconnects connected client on the server.

`virtual void Shutdown() = 0;`

Immediately shuts down all connections to server, and performs cleanup. It waits/blocks until function returns.

`virtual ClientData** GetClients() const = 0;`

Return value:
- A pointer to the array of clients.

`virtual USHORT ClientCount() const = 0;`

Return value:
- Returns the number of connected clients.

`virtual void Ping() = 0;`

Pings all clients connected to server.

`void Ping(Socket client);`

Pings the specified client.

`virtual void SetPingInterval(float interval) = 0;`

Sets the server's ping interval.

`virtual float GetPingInterval() const = 0;`

Return Value:
- Current value of ping interval.

`virtual bool MaxClients() const = 0;`

Return value:
- True:
  - If number of connected clients is at the maximum number of clients.

- False:
  - If number of connected clients is less than maximum clients.

virtual bool IsConnected() const = 0;

Return value:
- True:
  - If listening socket has been binded.
- False:
  - If listening socket has not been binded.

virtual Socket& GetHost() = 0;

Returns a reference to the connected socket.

Return value:
- Socket&

virtual void* GetObj() const = 0;

Returns a pointer to the object you specified in constructor.

Return value:
- Void*

Other:

CAMSNETLIB void WaitAndCloseHandle(HANDLE& hnd);

Waits for the specified handle to be triggered, then closes the specified handle.

Server and Client auto handled messages

Key:

-Checkmarks mean auto handled.

TYPE	MESSEAGE	SERVER	CLIENT	ADDITIONAL DATA

TYPE_PING (0)	MSG_PING (0)	Sent to client every X seconds. ✓	Sent to server every X seconds. ✓	NONE
TYPE_CHANGE (-128)	MSG_CHANGE_SERVERFULL (-128)	Sent to client when server is full. ✓	Should be handled in msgHandler on client.	NONE
TYPE_CHANGE (-128)	MSG_CHANGE_DISCONNECT (-127)	Sent to all clients when any user/pc disconnects ✓	Optionally handled on client.	std::wstring

Notice:

THIS IS A UNICODE BUILD ATTEMPTS TO USE MULTIBYTE/ASCII WILL RESULT IN A CRASH.

MsgStream

MsgStream(char* data, UINT capacity)

Parameters:
- data – pointer to allocated data.
- Capacity – capacity of allocated data.

char GetType() const

Return value:
- Type

char GetMsg() const

Return value:
- Msg

UINT GetSize() const

Return value:
- Capacity of stream.

UINT GetDataSize() const

Return value:
- Capacity of stream excluding MSG_OFFSET.

template<typename T> std::enable_if_t<std::is_arithmetic<T>::value, T>
operator[](UINT index) const

Return value:
- Returns data at position index, of type of T.

bool End() const

Return value:
- True:
  - If you have reached end of stream.
- False:
  - If you have not reached end of stream.

MsgStreamWriter

MsgStreamWriter(char type, char msg, UINT capacity)

Parameters:
- type – type of msg.
- msg - msg
- capacity – capacity of allocated data.

template<typename T> void Write(const T& t)
template<typename T> MsgStreamWriter& operator<<(const T& t)

Writes data referenced by t to stream.

template<typename T>
std::enable_if_t<std::is_arithmetic<T>::value> Write(T* t, UINT count)

Writes count data of type T of data pointed to by t.

template<typename T>
std::enable_if_t<std::is_arithmetic<T>::value> WriteEnd(T* t)

Writes data pointed to by t, until end of stream is reached.

```
template<typename... T>
static UINT SizeType()
```

Returns the sum of the size of the types passed(arithmetic types only).

```
template<typename... T>
static UINT SizeType(const T&... t)
```

Returns the sum of the size of the objects passed.

MsgStreamReader

```
MsgStreamReader(char* data, UINT capacity)
```

Parameters:
- data – pointer to data, obtained from msgHandler
- capacity – capacity of allocated data, obtained from msgHandler.

```
template<typename T> T Read()
```

Return value:

- Value of type T read in from stream.

```
template<typename T> void Read(T& t)
```

```
template<typename T> MsgStreamReader& operator>>(T& dest)
```

Reads data in from stream.

```
template<typename T>
std::enable_if_t<std::is_arithmetic<T>::value, T>* Read(UINT count)
```

Reads count T from stream and returns a pointer to this data.

```
template<typename T>
std::enable_if_t<std::is_arithmetic<T>::value, T>* ReadEnd()
```

Reads T until end of stream and returns a pointer to this data.

Defining own MsgStream specializations

The message stream classes cannot function correctly for any object/class that contains a pointer, or reference. This is due to the fact that pointers and references are relative to that instance of the program, and cannot be transferred. For this reason, Read, Write, WriteEnd, ReadEnd, operator[], and SizeType<typename…>() are restricted to arithmetic types.

However dispite this limitation, you are able to freely specialize these functions to suit your needs through the use of class specialization. The system is setup as follows:

Let X represent your typename. Note that X may be a templated type, and can substitute with template parameters.

You define the following class and the following functions:

```cpp
class StreamWriter::Helper<X> : public HelpBase<X>
{
public:
      Helper(StreamWriter& stream) : HelpBase(stream){}
      void Write(const X& t);
      static UINT SizeType(const X& t);
}
class StreamReader::Helper<X> : public HelpBase<X>
{
public:
      Helper(StreamReader& stream) : HelpBase(stream){}
      X Read();
};
```

The rest of the functions are simply wrappers around these functions such as operator<< or operator>>.

Within these functions you may use stream to access member functions of StreamReader and StreamWriter. This includes other class specializations. Note: you can also call functions from Helper<T>, though it is recommended you go through stream.

There are plenty of examples of MsgStream specialization in the streamext.h.

Examples

Note the following is pseudo code, and should only be used to understand the framework.

Client:

-Creating a basic client:

```cpp
void MsgHandler(TCPClientInterface& clint, const BYTE* data, DWORD nBytes, void* obj)
{
        char* dat = (char*)(&data[MSG_OFFSET]);
        nBytes -= MSG_OFFSET;
        MsgStreamReader streamReader((char*)data, nBytes);
        const char type = streamReader.GetType(), msg = streamReader.GetMsg();

        switch (type)
        {
                case TYPE_PING:
                {
                        switch(msg)
                        {
                        case MSG_PING:
                                clint.Ping();
                                break;
                        }
                        break;
                }//TYPE_PING
                case TYPE_CHANGE:
                {
                        switch(msg)
                        {
                        case MSG_CHANGE_SERVERFULL:
                        {
                                //Notify user server is full
                                break;
                        }

                        case MSG_CHANGE_DISCONNECT:
                        {

                                //Notify user server is a client has disconnected
                                break;
                        }
                        }
                }
                break;

                // Handle other cases

        }

}
```

```cpp
void DisconnectHandler(bool unexpected) // for disconnection
{
        if(unexpected)
        {
                // Notify user they have been disconnected
        }

        // Most likely do nothing because you caused the disconnection
}

InitializeNetworking();

TCPClientInterface* client = CreateClient(&MsgHandler, &DisconnectHandler);
bool res = client->Connect(L"ip", L"port number");
if (res)
{
        res = client->RecvServData();
        if (res)
        {
                //Ready to send packets
        }
}

CleanupNetworking();
```

-Sending packets from client to server:

```cpp
//Sends the number 5 to the server, excluding msg_type, and msg
int number = 5;
client->SendServData((char*)&number, sizeof(int));
```

or

```cpp
//Sends the number 5 to the server, including msg_type, and msg
MsgStreamWriter streamWriter(TYPE_, MSG_TYPE_, StreamWriter::SizeType<int>());
streamWriter.Write(number);
HANDLE hnd = client->SendServData(streamWriter, streamWriter.GetSize());
WaitAndCloseHandle(hnd);
```

Server:

-Creating a basic server:

```cpp
void DisconnectHandler(ClientData* data)
{
        //Do whatever possibly log disconnections?
}

void ConnectHandler(ClientData* data)
```

```cpp
{
      //Do whatever possibly log connections?
}



//Handles all incoming packets
void MsgHandler(TCPServInterface& serv, ClientData* const clint, const BYTE* data, DWORD
nBytes, void* obj)
{
      auto clients = serv.GetClients();
      const USHORT nClients = serv.ClientCount();

      char* dat = (char*)(&data[MSG_OFFSET]);
      nBytes -= MSG_OFFSET;
      MsgStreamReader streamReader((char*)data, nBytes);
      const char type = streamReader.GetType(), msg = streamReader.GetMsg();

      //Switch type and msg for all your packets

}

InitializeNetworking();

//Optional port map on router
MapPort(port, L"TCP", L"Server");
TCPServInterface* serv = CreateServer(&MsgHandler, &ConnectHandler, &DisconnectHandler);
bool res = serv->AllowConnections(L"port");
if (res)
{
      //Ready to send packets
}

CleanupNetworking();
```

-Sending packets from server to client:

```cpp
//Sends the number 5 to all clients on server, excluding msg_type, and msg
int number = 5;
serv->SendClientData((char*)&number, sizeof(int), Socket(), false);
```

Or

```cpp
//Sends the number 5 to only the pc you specified, including msg_type, and msg
int number = 5;
MsgStreamWriter streamWriter(TYPE_, MSG_TYPE_, StreamWriter::SizeType<int>());
streamWriter.Write(number);
HANDLE hnd = serv->SendClientData(streamWriter, streamWriter.GetSize(), socket, true);
WaitAndCloseHandle(hnd);
```