**Lab Report**

**Lab 8 - Networking Lab 02**

**Craig Broskow**

**October 28, 2015**

Introduction:

This lab started experimenting with Low-Level API (LLAPI) network programming with Unity. As such, it concentrated on basic socket management and message sending over a network.

Methods:

The first task in this lab was to create a basic server socket. This socket operates at the Transport layer of the OSI model and allows network connections from other computers. Since Unity's UNET networking technology was not used in this lab, we had to store all networking data (such as socket and channel IDs) in local variables within our code. One of the essential pieces of information which we worked with was the GlobalConfig, which stores basic network information such as maximum packet size and the length of the message queue. It is essential that the GlobalConfig data must be the same for the server computer and client computers which connect to it...or the network connections will fail.

Another data set that is required for reliable and unreliable communication over the network is called the HostTopology. That data structure contains information regarding TCP and UCP channel configurations and also the maximum number of connections that the server will support. Together, the GlobalConfig and the HostTopology data structures (along with a port number assignment) are sufficient to initialize the network and open a socket that enables network connections between computers.

Network messages are stored in a queue which is accessed by assignment of the NetworkEventType.DataEvent data structure. That queue was iterated through in a while loop, and the individual types of messages (such as connections, disconnects and data messages) were handled on an individual basis via a switch statement. On the Server side, connections and disconnects were merely noted. Data messages were serialized (so that Unity could interpret them) and processed using a MemoryStream (for speed) and a BinaryFormatter for serializing and deserializing the data streams. Methods were written for both reading and writing network messages.

Finally, Client-side code was written which closely mirrored the code written for the Server side except that the Client did not need to keep track of the connections to the Server.

Conclusions:

The code in this lab was certainly more complicated and "closer to the metal" than in previous labs. The LLAPI technology used was very unforgiving and required strict adherence to the type and sequence of commands expected.

It's not known why, but for some reason my Client code never successfully connected to the Server code. At least superficially, the Server code appeared to be working correctly...as did the Client code when it was running in the Unity Editor and not in a stand-alone executable. I'm hopeful that this failure is only due to some incorrect parameters and not a fundamental misunderstanding of the data structures and commands necessary to implement networking within Unity using the LLAPI.

Postlab Questions:

1. What does the connectionID mean when the client receives messages from a server?

In this case, connectionID refers to a unique identifier for the server computer which sent the message.

Code:

// ServerConnection.cs

using UnityEngine;

using System.Collections;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

using UnityEngine.Networking;

public class ServerConnection : MonoBehaviour {

int serverSocketID = -1;

int maxConnections = 10;

byte unreliableChannelID;

byte reliableChannelID;

bool serverInitialized = false;

// Use this for initialization

void Start () {

DontDestroyOnLoad(this);

GlobalConfig globalConfig = new GlobalConfig ();

globalConfig.ReactorModel = ReactorModel.FixRateReactor;

globalConfig.ThreadAwakeTimeout = 10;

ConnectionConfig connectionConfig = new ConnectionConfig ();

reliableChannelID = connectionConfig.AddChannel (QosType.ReliableSequenced);

unreliableChannelID = connectionConfig.AddChannel (QosType.UnreliableSequenced);

HostTopology hostTopology = new HostTopology (connectionConfig, maxConnections);

NetworkTransport.Init(globalConfig);

serverSocketID = NetworkTransport.AddHost (hostTopology, 7777);

if (serverSocketID < 0)

Debug.Log ("Server socket creation failed!");

else

Debug.Log ("Server socket creation success");

serverInitialized = true;

}

// Update is called once per frame

void Update ()

{

if(!serverInitialized)

return;

int recHostId;

int connectionId;

int channelId;

int dataSize;

byte[] buffer = new byte[1024];

byte error;

NetworkEventType networkEvent = NetworkEventType.DataEvent;

do

{

networkEvent = NetworkTransport.Receive( out recHostId , out connectionId , out channelId , buffer , 1024 , out dataSize , out error );

switch(networkEvent)

{

case NetworkEventType.Nothing:

break;

case NetworkEventType.ConnectEvent:

if( recHostId == serverSocketID )

{

Debug.Log ("Server: Player " + connectionId.ToString () + " connected!" );

}

break;

case NetworkEventType.DataEvent:

if( recHostId == serverSocketID )

{

Stream memoryStream = new MemoryStream(buffer);

BinaryFormatter binaryFormatter = new BinaryFormatter();

string message = binaryFormatter.Deserialize( memoryStream ).ToString ();

Debug.Log ("Server: Received Data from " + connectionId.ToString () + "! Message: " + message );

RespondMessage(message, connectionId);

}

break;

case NetworkEventType.DisconnectEvent:

if( recHostId == serverSocketID )

{

Debug.Log ("Server: Received disconnect from " + connectionId.ToString () );

}

break;

}

} while ( networkEvent != NetworkEventType.Nothing );

}

void SendMessage(string message, int target)

{

byte error;

byte[] buffer = new byte[1024];

Stream memoryStream = new MemoryStream(buffer);

BinaryFormatter binaryFormatter = new BinaryFormatter();

binaryFormatter.Serialize (memoryStream, message);

NetworkTransport.Send (serverSocketID, target, reliableChannelID, buffer, (int)memoryStream.Position, out error);

if (error != (byte)NetworkError.Ok)

{

NetworkError networkError = (NetworkError) error;

Debug.Log ("Error: " + networkError.ToString ());

}

}

void RespondMessage(string message, int playerID)

{

if (message == "FirstConnect")

{

Debug.Log ("message was FirstConnect! from player " + playerID.ToString());

SendMessage("goto\_NewScene", playerID);

if (Application.loadedLevelName != "Scene2")

Application.LoadLevel("Scene2");

}

}

}

// ClientConnection.cs

using UnityEngine;

using System.Collections;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

using UnityEngine.Networking;

public class ClientConnection : MonoBehaviour {

int clientSocketID = -1;

//Will store the unique identifier of the session that keeps the connection between the client

//and the server. You use this ID as the 'target' when sending messages to the server.

int clientServerConnectionID = -1;

int maxConnections = 10;

byte unreliableChannelID;

byte reliableChannelID;

bool isClientConnected = false;

void Start()

{

DontDestroyOnLoad(this);

//Build the global config

GlobalConfig globalConfig = new GlobalConfig ();

globalConfig.ReactorModel = ReactorModel.FixRateReactor;

globalConfig.ThreadAwakeTimeout = 10;

//Build the channel config

ConnectionConfig connectionConfig = new ConnectionConfig ();

reliableChannelID = connectionConfig.AddChannel (QosType.ReliableSequenced);

unreliableChannelID = connectionConfig.AddChannel (QosType.UnreliableSequenced);

//Create the host topology

HostTopology hostTopology = new HostTopology (connectionConfig, maxConnections);

//Initialize the network transport

NetworkTransport.Init(globalConfig);

//Open a socket for the client

clientSocketID = NetworkTransport.AddHost (hostTopology, 7777);

//Make sure the client created the socket successfully

if (clientSocketID < 0)

Debug.Log ("Client socket creation failed!");

else

Debug.Log ("Client socket creation success");

//Create a byte to store a possible error

byte error;

//Connect to the server using

//int NetworkTransport.Connect(int socketConnectingFrom, string ipAddress, int port, 0, out byte possibleError)

//Store the ID of the connection in clientServerConnectionID

clientServerConnectionID = NetworkTransport.Connect(clientSocketID, "127.0.0.1", 7777, 0, out error);

//Display the error (if it did error out)

if (error != (byte)NetworkError.Ok)

{

NetworkError networkError = (NetworkError) error;

Debug.Log ("Error: " + networkError.ToString ());

}

}

void Update()

{

//If the client failed to create the socket, leave this function

if (clientSocketID < 0)

return;

PollBasics();

//If the user pressed the Space key

//Send a message to the server "FirstConnect"

if (Input.GetKey(KeyCode.Space))

SendMessage("FirstConnect");

//If the user pressed the R key

//Send a message to the server "Random message!"

if (Input.GetKey(KeyCode.R))

SendMessage("Random message!");

}

void SendMessage(string message)

{

//create a byte to store a possible error

byte error;

//Create a buffer to store the message

byte[] buffer = new byte[1024];

//Create a memory stream to send the information through

Stream memoryStream = new MemoryStream(buffer);

//Create a binary formatter to serialize and translate the message into binary

BinaryFormatter binaryFormatter = new BinaryFormatter();

//Serialize the message

binaryFormatter.Serialize (memoryStream, message);

//Send the message from this client, over the client server connection, using the reliable channel

NetworkTransport.Send (clientSocketID, clientServerConnectionID, reliableChannelID, buffer, (int)memoryStream.Position, out error);

//Display the error (if it did error out)

if (error != (byte)NetworkError.Ok)

{

NetworkError networkError = (NetworkError) error;

Debug.Log ("Error: " + networkError.ToString ());

}

}

void InterperateMessage(string message)

{

//if the message is "goto\_NewScene"

//load the level named "Scene2"

if (message == "goto\_NewScene")

Application.LoadLevel("Scene2");

}

void PollBasics()

{

//prepare to receive messages by practicing good bookkeeping

int recHostId;

int connectionId;

int channelId;

int dataSize;

byte[] buffer = new byte[1024];

byte error;

NetworkEventType networkEvent = NetworkEventType.DataEvent;

//do

do

{

//Receive network events

networkEvent = NetworkTransport.Receive( out recHostId , out connectionId , out channelId , buffer , 1024 , out dataSize , out error );

//switch on the network event types

switch(networkEvent)

{

//if nothing, do nothing

case NetworkEventType.Nothing:

break;

//if connection

case NetworkEventType.ConnectEvent:

//verify that the message was meant for me

if (recHostId == clientSocketID)

{

//debug out that i connected to the server, and display the ID of what I connected to

Debug.Log ("Client: Server " + connectionId.ToString () + " connected!" );

}

//set my bool that is keeping track if I am connected to a server to true

isClientConnected = true;

break;

//if data event

case NetworkEventType.DataEvent:

//verify that the message was meant for me and if I am connected to a server

if ((recHostId == clientSocketID) && isClientConnected)

{

//decode the message (bring it through the memory stream, deseralize it, translate the binary)

Stream memoryStream = new MemoryStream(buffer);

BinaryFormatter binaryFormatter = new BinaryFormatter();

string message = binaryFormatter.Deserialize( memoryStream ).ToString ();

//Debug the message and the connection that the message was sent from

Debug.Log("Client: Received Data from " + connectionId.ToString () + "! Message: " + message);

InterperateMessage(message); //the message to interperate

}

break;

//if disconnection

case NetworkEventType.DisconnectEvent:

//verify that the message was meant for me, and that I am disconnecting from the current connection I have with the server

if ((recHostId == clientSocketID) && isClientConnected)

{

NetworkTransport.Disconnect(clientSocketID, connectionId, out error);

//debug that I disconnected

Debug.Log("Client: Disconnected from " + connectionId.ToString());

//set my bool that is keeping track if I am connected to a server to false

isClientConnected = false;

}

break;

}

//while (the network event I am receiving is not Nothing)

} while ( networkEvent != NetworkEventType.Nothing );

}

}