CMP303/501 Network (Systems for) Game Development

Lab 1: Basic TCP Sockets

Andrei Boiko, Laith Al-Jobouri (and Gaz Robinson)
School of Design and Informatics

Introduction

In this lab, we're going to experiment with a simple client-server system where two programs are connected using TCP sockets. In particular, we're going to try simulating some of the common mistakes and network problems that you'll probably encounter later when programming with sockets.

If you have any questions or problems with this week's practical, ask us on-campus or in the MS Teams channel for "Lab Help". If you have a general question about the module, please use the "General" channel or email us (a.boiko@abertay.ac.uk, l.al-jobouri@abertay.ac.uk).

The application

Get Lab_1_TCP_Sockets.zip from the Week 1 page on MyLearningSpace, and unpack it. It contains two projects, client and server. Open both projects in **Visual Studio** and compile them.

The server program is an "echo server": it accepts messages from the client, and just sends them back to the client immediately. The client program reads a message from the user, sends it to the server, and prints out whatever it gets back. To test the system out, first start the server program, then start the client. Both programs print various debugging messages as they run so you can follow along in the source code.

You will want to have the WinSock documentation available; search for `WSAStartup` on msdn.microsoft.com.

Things to try (in order)

- Read and consider the code. There are similar sections in the two programs; compare the two and identify the differences.
- The sockets API requires port numbers to be stored in "network byte order"; the `ntohs` and `htons` functions are used to convert to and from this. What happens if you remove them from the client (only)?
- What happens (specifically, in terms of error responses from functions) if you run the client without first running the server?
    - ... or if you stop the server while the client is still running, or vice versa?
- What happens if you forget to call listen or bind in the server? (Can you think of why this might have been separated into two different functions? We'll discuss this later.)

Continued on the next page…

- Error handling is important. Both programs have “FIXME” comments suggesting places where they can be made more robust – add the appropriate checks.
    - To work out what to check for, look up each sockets function on MSDN and read the section about what its return value means. It's often helpful to think about what the return value should be when the call is successful. Make sure you don't accidentally end up calling the function twice when you write the check!
- What happens if you send a really long message, and why? The programs don't handle this very well at the moment... come up with a fix for this.
- Make the server send back a capitalised (or reversed, or similar) version of the message.
- Try making the client bind to a specified port number