CMP303/501 Network (Systems for) Game Development

# Lab 5: Event-Based I/O

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

## Introduction

In this lab, we'll experiment with event-based I/O using `WSAEventSelect`.

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 "EventServer.zip" from MyLearningSpace and unpack it. It contains one working Visual Studio project, EventServer.

This is a really basic TCP server. It has the ability to accept one connection and gracefully allow it to close… and that's it! The kicker is that it manages this functionality with Event Objects!

You'll see that there are two Sockets (one for the listener, and one for the accepted client) and two matching event objects (again, one for events on the listener, one for events on the client).

Here, whenever we set up a socket, we need to remember to also create a corresponding event, which we do with `WSACreateEvent()`, and on top of that we must also call `WSAEventSelect` to associate this new event with any network messages that the socket receives.

If you run the program, and run any of the TCP clients from previous labs, you'll see it accept the incoming connection and build a new event for the socket. If the client quits, the server closes the connection because the `EventSelect` has been told to look out for `FD_CLOSE` events.

## Improving the server

Right now the server is incredibly limited; it doesn't even receive data! Lets work at improving it!

- The server should be able to handle multiple connections at a time. Design and implement a system that allows for more than one client to connect to the server at any one time.
- Remember, each socket will need to have a corresponding event!
- Make the server able to receive and send messages.
- Right now the accepted socket only listens our for an `FD_CLOSE` event. Set up EventSelect so that the corresponding event is signalled when `FD_READ` or `FD_WRITE` are available.
- Remember: These functions are only advisory. You must be prepared to receive a `WSAWOULDBLOCK` error
- Look again at the Connection class from last week's lab; it provided a way to encapsulate the operation of a connection (managing the socket, socket state, and send/receive buffers). Build the class, or something equivalent, in this week's project.

## Building an EventSelect-based client

Once you have a working `EventSelect` server, take what you've learned and apply the knowledge to the client side!

## UDP functionality

We're all aware of the power that UDP has when it comes to real-time networking, so see if you can modify the server and client to make use of UDP sockets using `EventSelect`.

## Assistance

It's recommended you have the documentation for the functions we're working with open on MSDN. It has good summaries of what values you can expect, how you should approach implementation and code examples for some pieces. The rules for some of these practices are fairly complex; please discuss them with us if you're not sure.

Some hints:

- Make sure when you read or write, that you're reading/writing all the data available.
    - If there's a backlog of messages and you're still only reading one per tick, you might end up behind in the queue.
- FD_READ tends to fire immediately, as the socket is immediately available for writing. Be sure you don't try and send every time WRITE is available. Have another mechanism, like the Connection class's state machine, to help manage reading and writing.