FALMOUTH UNIVERSITY

Lecture 1: Module Introduction

COMP260: Distributed Systems
BSc (Hons) Computing for Games

- Today’s session:
  - Introduction to the module & assignments
  - Introduction to socket-based programming

- Introduction to the module

- Introduction to the module
  - A 10 week module to introduce you to distributed processing (networking)
  - In two parts:
    - My part (turn-based networking)
      - Fundamental socket programming
      - Technical Architecture
      - Hosting services on remote servers
    - Al’s part (real-time networking)
      - Real-time provision in Unity & networking engines
      - Games that rely on object duplication & synchronisation

- Introduction to the module

Week 1	Week 2	Week 3	Week 4	Week 5	Reading Week
	Turn-based Service Provision				
Introduction	IP & Socket Programming	Networking & Concurrency	Concurrency in Clients (chat service)	Games-As-A-Service (hosting)	
	Proposal Review		Tutorial		Tutorial
	Portfolio Dev	Portfolio Dev	Portfolio Dev	Portfolio Dev	

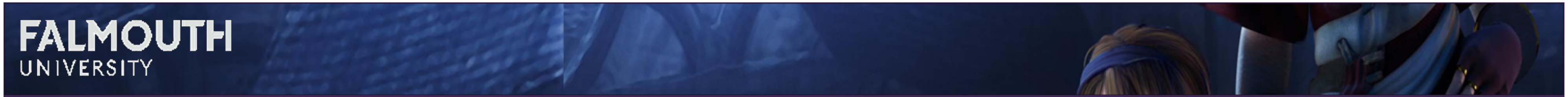
Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Real-time Gaming Provision					
Real-time Networking 1	Real-time Networking 2	Real-time Networking 3	Real-time Networking 4		
	Tutorial				
Portfolio Dev	Portfolio Dev	Portfolio Dev	Peer Review	Portfolio Dev	Portfolio Dev

Week 13
VIVA OF DOOM

- Introduction to the module
  - Assignments
    - Assignment 1: Computing Artefact
      - Two Parts
        - Create a turn-based game & host on a remote server
          - Technical analysis
          - Technical design
          - Demo on remote server
        - Create a real-time networked game
          - Create a ‘simple’ game that multiple people can play together over a network
            - FPS Deathmatch
            - Multiplayer arcade game
    - Assignment 2: Technical Report
      - This is shared across all individual specialist computing projects modules

- Introduction to the module

- Do you have any questions for me?

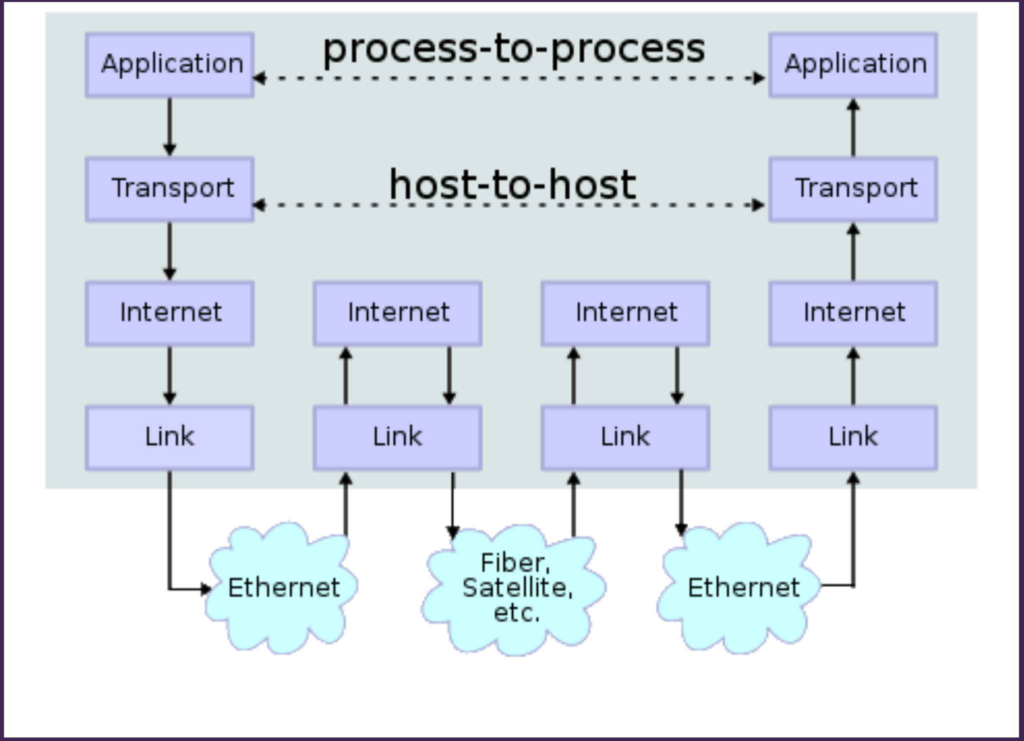


- Introduction to socket-based programming

- Introduction to socket-based programming
  - In COMP280, we looked at HTTP as a network protocol
    - POST & GET provide client-controlled communications
    - Network gaming requires server to initiate communication with client, so HTTP is no use

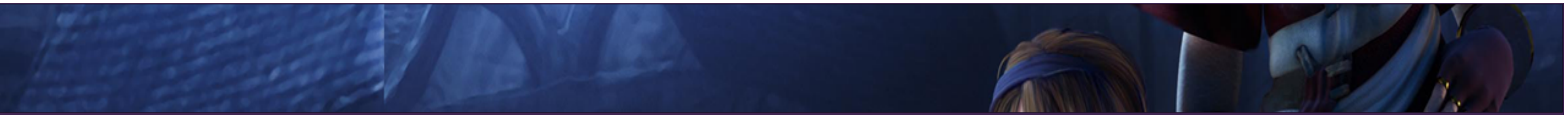
- Introduction to socket-based programming
  - In COMP280, we looked at HTTP as a network protocol
    - HTTP is part of the internetworking protocol (IP) stack
      - Along with all the other ‘TP’ services we use (FTP, SMTP etc)
    - The underlying networking stack for IP defines end-to-end communications over connected networks (inter networking)
      - This comes from the US Dept. of Defence cold-war research for nuclear-proof networking

- Introduction to socket-based programming
  - IP Stack



From a software developer's perspective, an application using the IP to communicate works across platforms (they don't have to think about the underlying stack)

- Introduction to socket-based programming
  - IP Stack
    - Two flavours of IP stack we are interested in for games:
    - TCP/IP
      - Guaranteed delivery & guaranteed order of deliver
      - ‘slow n steady’
    - Datagram/IP
      - Nothing is guaranteed
      - ‘fast n loose’
    - Early network games (Quake et al) used datagrams as it allowed the most data to be sent and had no stalling (through retries)
      - Far more common to use TCP/IP nowadays



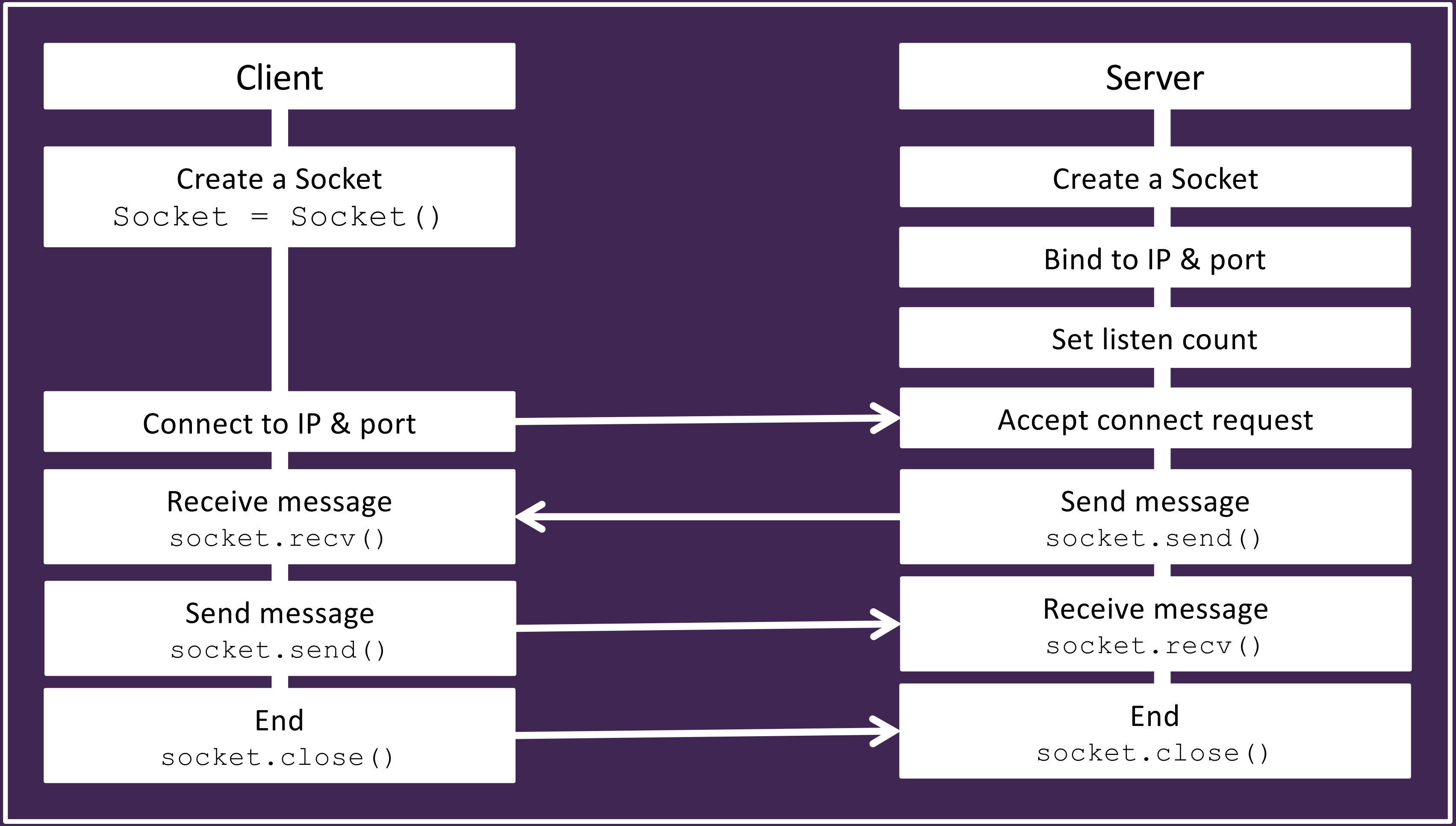
- Introduction to socket-based programming
  - IP Sockets
    - Host-to-host communications are implemented through sockets & socket libraries
    - Platform-independent
      - Sockets define a protocol (messages & data formats) any user of sockets has to implement that protocol
      - System / language interoperability
        - » Anything that uses sockets can communicate with anything else that uses sockets

- Introduction to socket-based programming
  - Anatomy of socket communications

Server Use Case	Client Use Case
• Create a socket • Listen for clients • Accept new connection(s) • Receive and send data • Close connections	• Create a socket • Connect to server • Receive and send data • Close connections

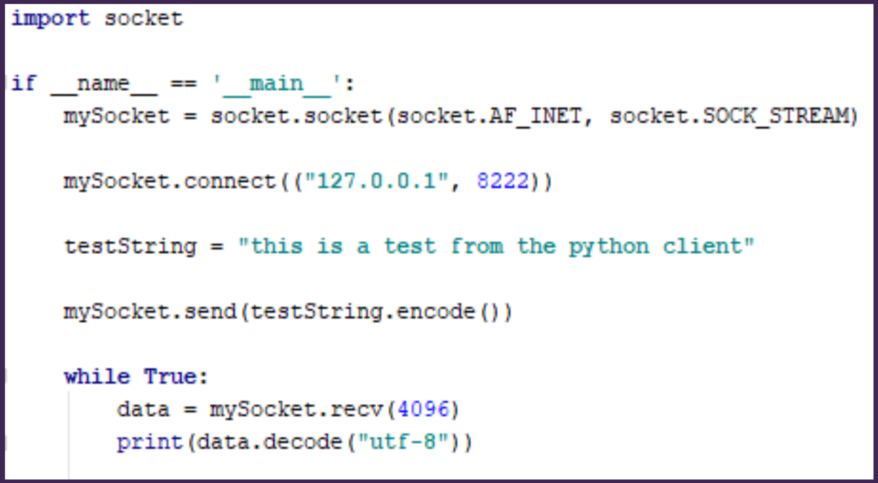
  - Server and client are roles, rather than bits of h/w
    - Server will serve up data for requests from clients
    - An app can be both a client and a server (to different servers and clients)

- Introduction to socket-based programming
  - Anatomy of socket communications



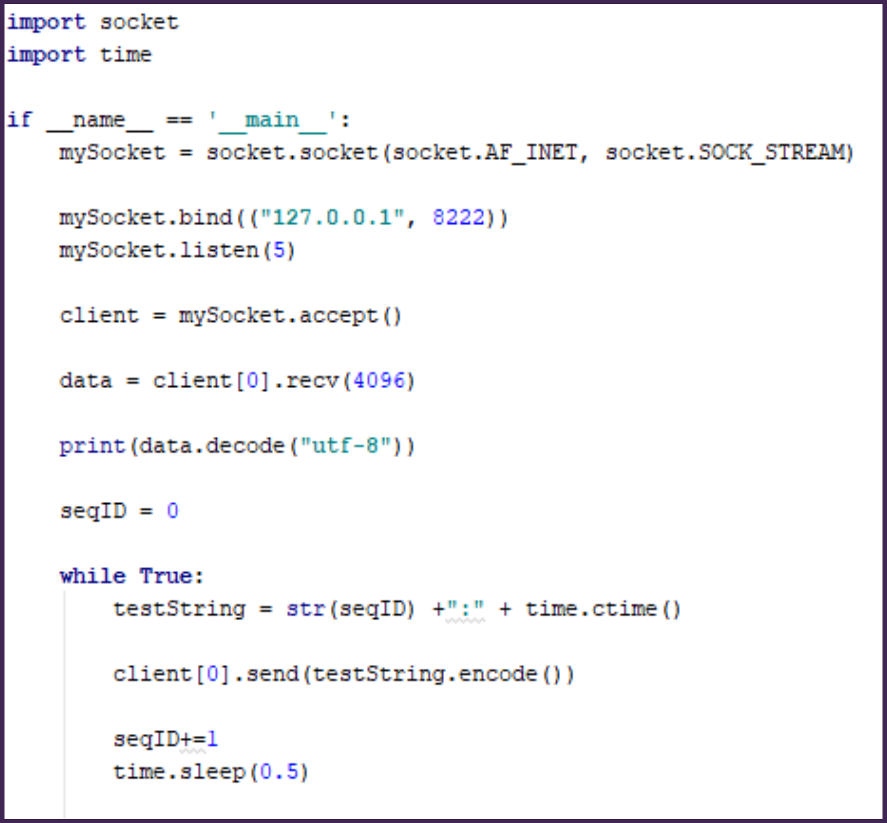
- Introduction to socket-based programming
  - In Python

Client



```
import socket

if __name__ == '__main__':
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    mySocket.connect(("127.0.0.1", 8222))

    testString = "this is a test from the python client"

    mySocket.send(testString.encode())

    while True:
        data = mySocket.recv(4096)
        print(data.decode("utf-8"))
```

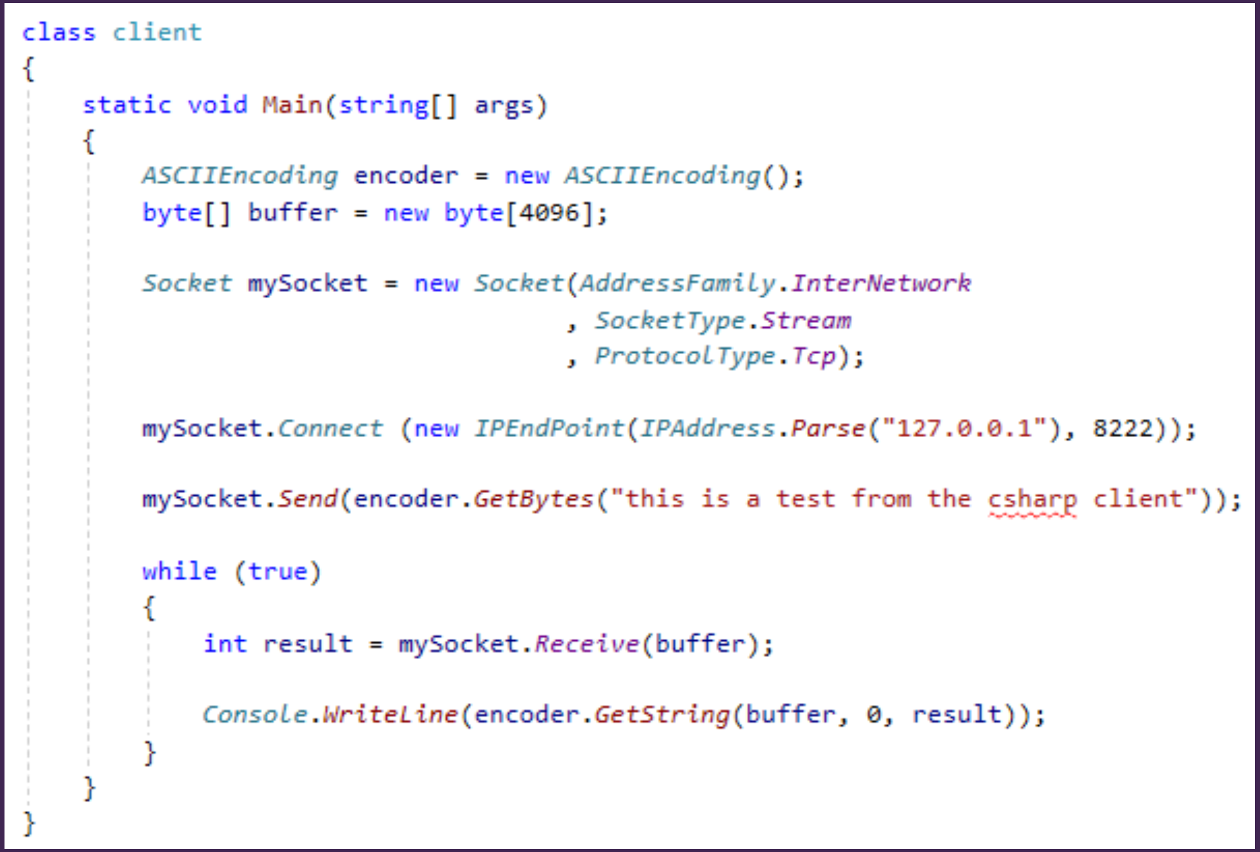
Server



```
import socket
import time

if __name__ == '__main__':
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    mySocket.bind(("127.0.0.1", 8222))
    mySocket.listen(5)

    client = mySocket.accept()

    data = client[0].recv(4096)

    print(data.decode("utf-8"))

    seqID = 0

    while True:
        testString = str(seqID) +":" + time.ctime()

        client[0].send(testString.encode())

        seqID+=1
        time.sleep(0.5)
```

- Introduction to socket-based programming
  - In C#

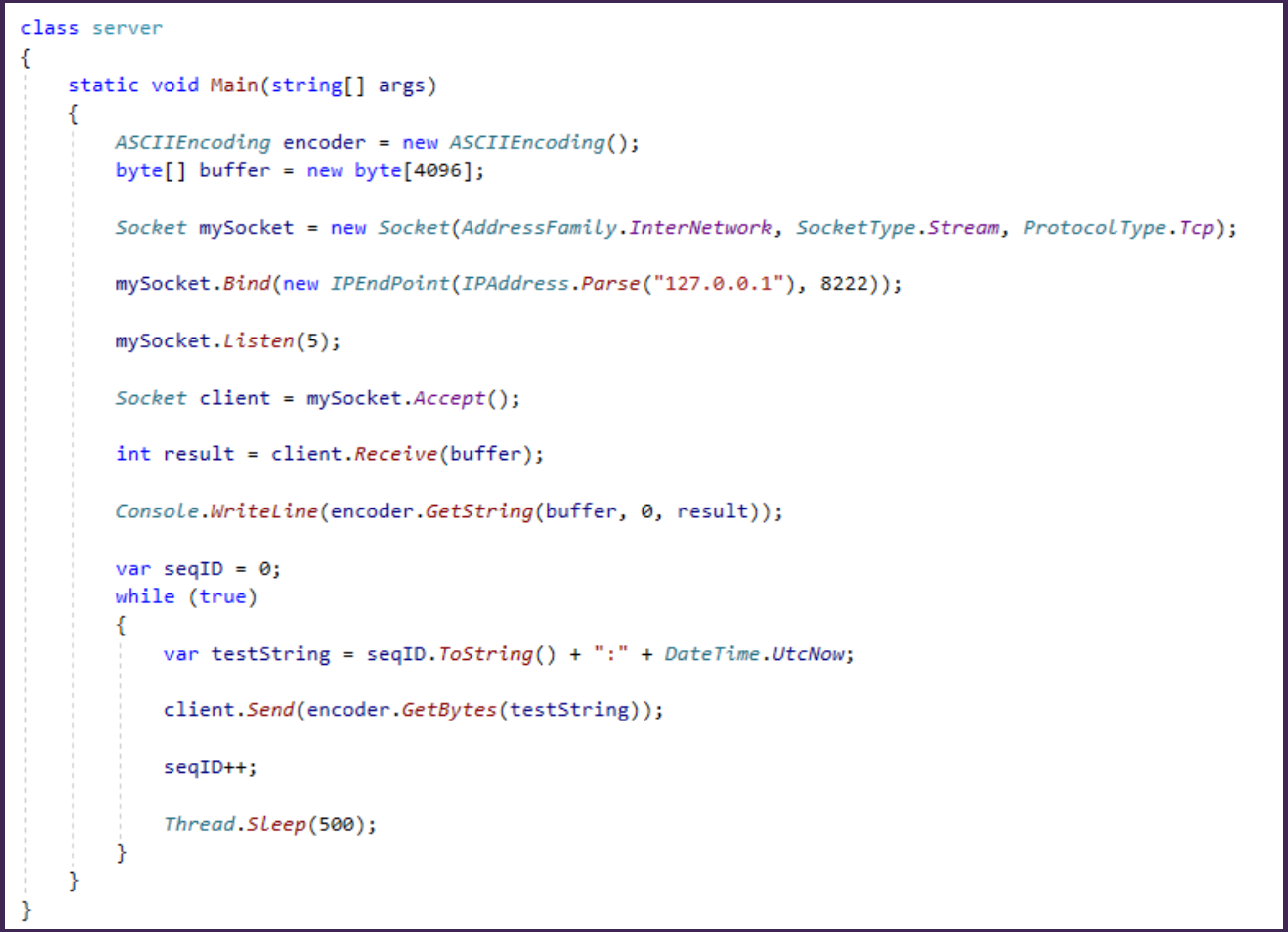
Client



```
class client
{
    static void Main(string[] args)
    {
        ASCIIEncoding encoder = new ASCIIEncoding();
        byte[] buffer = new byte[4096];

        Socket mySocket = new Socket(AddressFamily.InterNetwork
                                     , SocketType.Stream
                                     , ProtocolType.Tcp);

        mySocket.Connect (new IPEndPoint(IPAddress.Parse("127.0.0.1"), 8222));

        mySocket.Send(encoder.GetBytes("this is a test from the csharp client"));

        while (true)
        {
            int result = mySocket.Receive(buffer);

            Console.WriteLine(encoder.GetString(buffer, 0, result));
        }
    }
}
```

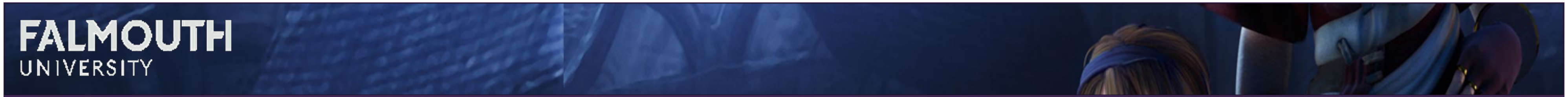
- Introduction to socket-based programming
  - In C#

Server



```
class server
{
    static void Main(string[] args)
    {
        ASCIIEncoding encoder = new ASCIIEncoding();
        byte[] buffer = new byte[4096];

        Socket mySocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

        mySocket.Bind(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 8222));

        mySocket.Listen(5);

        Socket client = mySocket.Accept();

        int result = client.Receive(buffer);

        Console.WriteLine(encoder.GetString(buffer, 0, result));

        var seqID = 0;
        while (true)
        {
            var testString = seqID.ToString() + ":" + DateTime.UtcNow;

            client.Send(encoder.GetBytes(testString));

            seqID++;

            Thread.Sleep(500);
        }
    }
}
```

- Introduction to socket-based programming
  - Regardless of programming language
    - Sockets work in the same way
      - Create, bind, listen, connect, accept, send receive, close
      - ‘Broadly’ equivalent to working with files (open, read, write, close)
        - Think of a socket as a file you can read & write to
    - Data is sent as a stream of bytes (just like HTTP)
      - Convert data to bytes (serialise)
      - Send it as bytes
      - Receive it as bytes
      - Convert bytes to data (de-serialise)
  - This makes it ‘easy’ to communicate between applications, machines and operating systems
    - Sockets are relatively platform agnostic
    - Be aware of string formats (ASCII encoding, utf-8 etc)



- Questions

- For Next Week
  - Experiment with the client/server code
    - Particularly mix & match Python and C# applications
  - Proposal Reviews on Tuesday
    - What do want to build with AI?