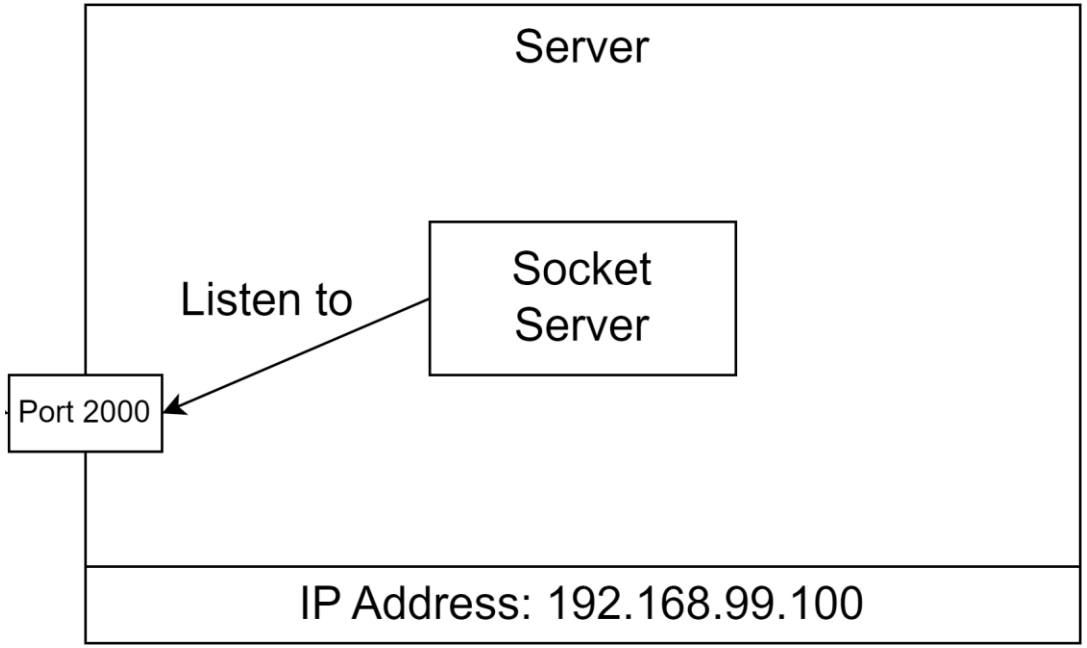
Client-Server communication in C#

AARHUS UNIVERSITY
AARHUS UNIVERSITY SCHOOL OF ENGINEERING

MICHAEL LOFT
ML@ASE.AU.DK

Socket communication

The server listens



The Server specifies where to listen:

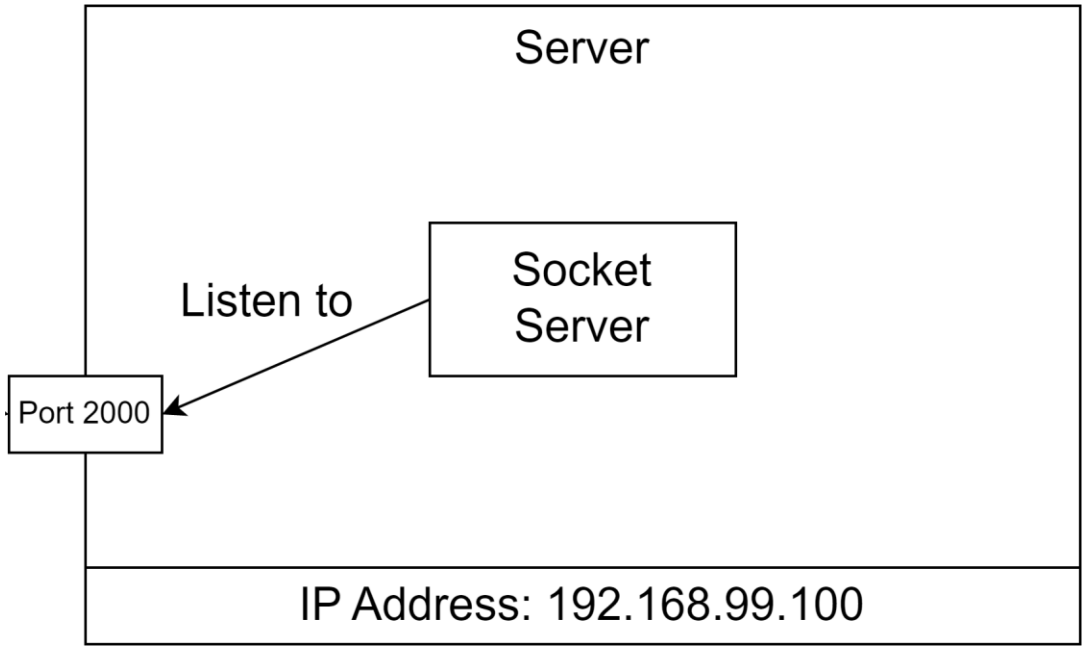
- IP Address
- Port number

The server listens

The server can have multiple network cards and more than one address.

IP Address to listen to can be:

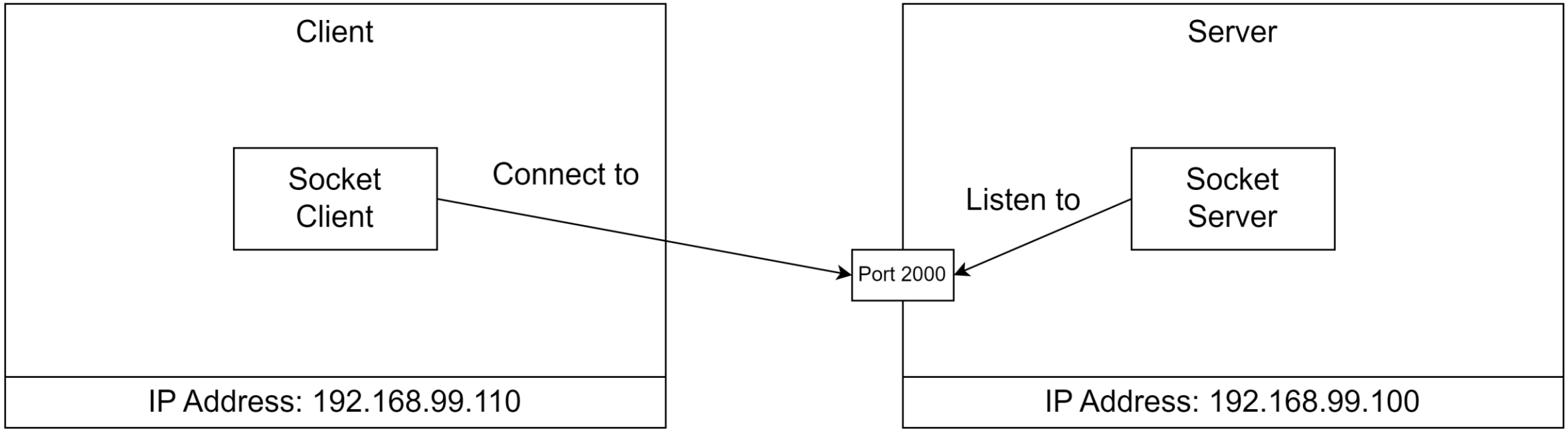
- **Any** <= all network interfaces
- **Loopback** <= 127.0.0.1
- A specific IP address



The Server specifies where to listen:

- IP Address
- Port number

The client connects to the server



The client connects to a server socket:

- IP Address
- Port number

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Create and IPEndPoint with Address = Any and Port = 2000.

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Create a new socket.

- ‘AddressFamily’ is the one for IP addresses – ”InterNetwork”.
- Socket type is Stream, which means a two-way data stream.
- The protocol type is Transmission Control Protocol - TCP.

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addres    for this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipA    ss, 2000);

        using Socket listener = new Socket(ipE    Point.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Co    ole.WriteLine($"Listening on: {ipAddress}");

                                                    ffer, SocketFlags.None);
                                                    uffer, 0, numberOfBytesReceived);
                                                    ta}");

            byte[] replyBytes = Encoding.Utf8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Create a new socket.

- ‘AddressFamily’ is the one for IP addresses – ”InterNetwork”.
- Socket type is Stream, which means a two-way data stream.
- The protocol type is Transmission Control Protocol - TCP.

The Socket implements the ‘IDisposable’ interface.

The ‘using’ keywork means, that the ‘listener’ object will be disposed correctly, even if any exceptions occur in the code.

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Associate socket with local endpoint on the machine.

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Start listening on the socket.

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses f  this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress  00);

        using Socket listener = new Socket(ipEndPoi  ddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {i  ress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Wait for a connection from a client.

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, S  ketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Read bytes from the socket in to a buffer

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, S  ketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(b  fer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Convert to text (UTF-8)

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, S ketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(b fer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Read bytes from the socket in to a buffer

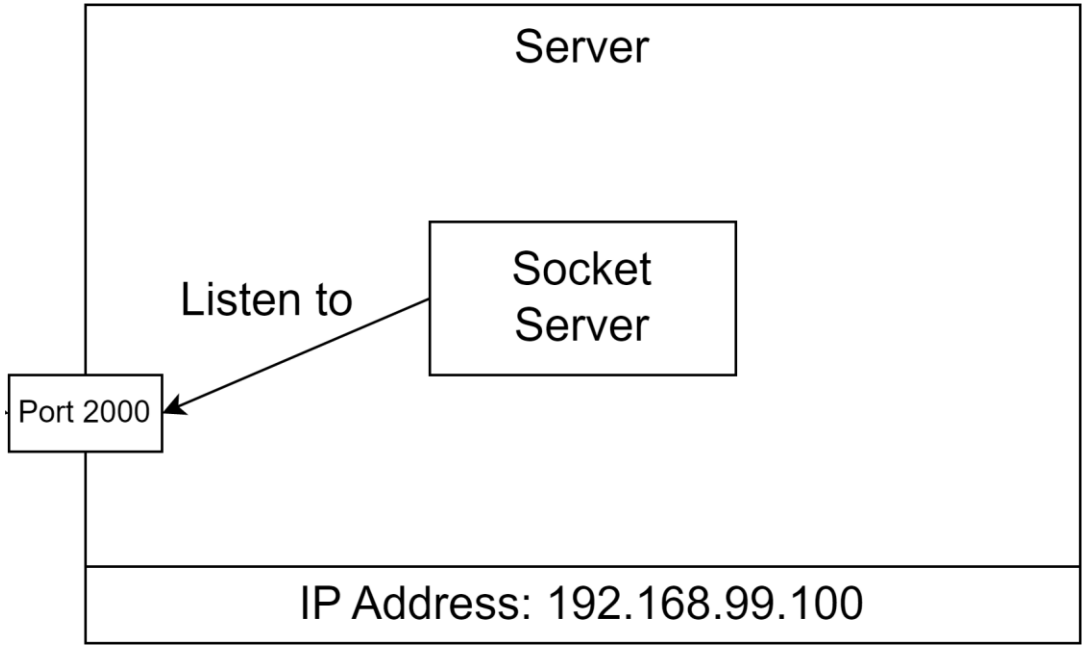
SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this m
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, S  ketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(b  fer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetStri  (buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{recei  Data}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Send ”ACK” message back to the client.

NOTE:
There is no requirement to send anything back to the client. It is only included here to show two-way communication.

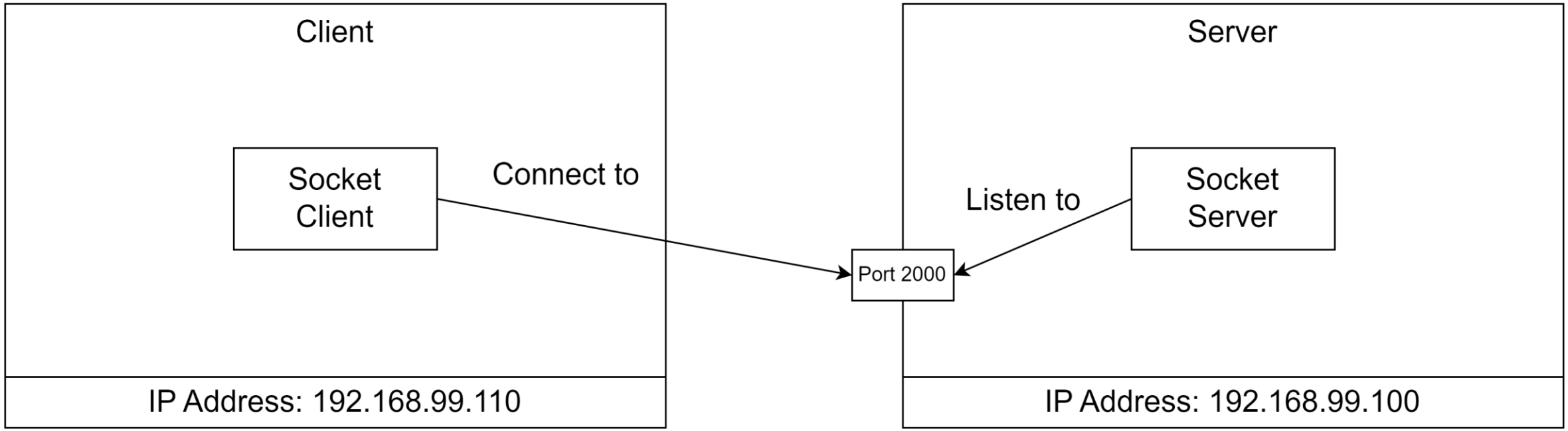
The server listens



The Server specifies where to listen:

- IP Address = Any
- Port number = 2000

The client connects to the server



The client connects to a server socket:

- IP Address
- Port number

SocketClient

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

SocketClient

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

Create and IPEndPoint with
Address = 127.0.0.1 (localhost)
and
Port = 2000.

If the server is on a different machine, the Address shall be the IP address of that machine.

SocketClient

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

Create the socket and connect to the IP Endpoint.

The connection attempt may time out if the endpoint does not exist, which will throw an exception.

SocketClient

Create the message to send.
Convert to bytes.
Send the bytes.

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 20[illegible];

        using Socket client = new Socket(ipEndPoint.Addr[illegible]Family, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

SocketClient

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

Receive ”ACK” message from the server.

NOTE:
There is no requirement to send anything back to the client. It is only included here to show two-way communication.

SocketClient

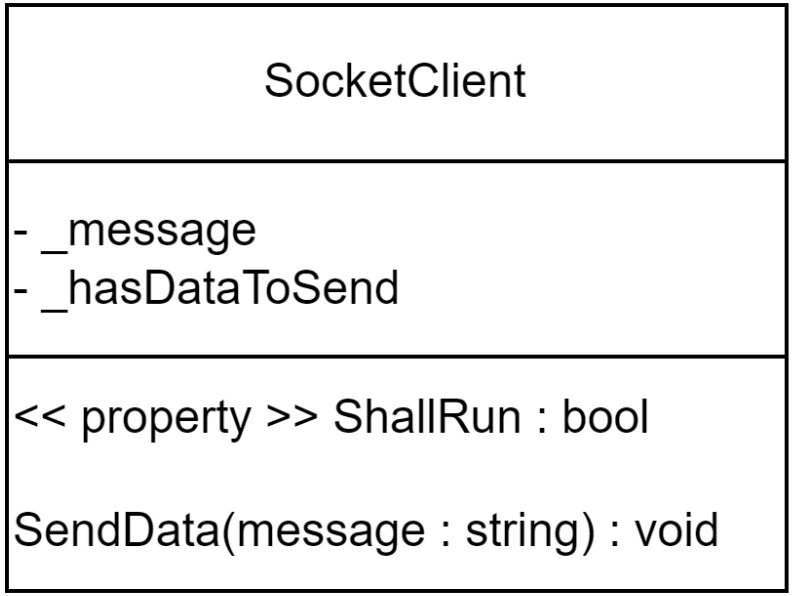
```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

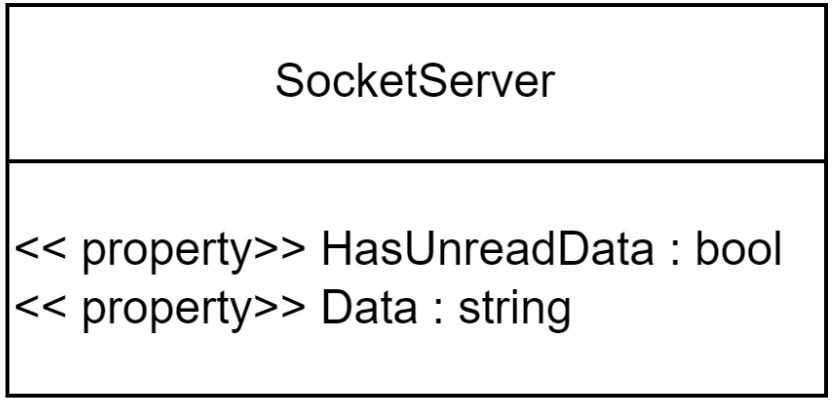
Shutdown both sending and receiving socket when done.

Your turn

Solve the ”Exercise - Client-Server communication”

Using the socket communication from other threads





Sending data from the client

```
internal class SocketClient
{
    private string _message = "";
    private bool _hasDataToSend = false;
    private bool _shallRun = true;

    public bool ShallRun { private get => _shallRun; set => _shallRun = value; }

    public void SendData(string message)
    {
        _message = message;
        _hasDataToSend = true;
    }

    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        while (ShallRun)
        {
            // Send message.
            if (_hasDataToSend)
            {
                var message = _message;
                var messageBytes = Encoding.UTF8.GetBytes(message);
                client.Send(messageBytes, SocketFlags.None);
                Console.WriteLine($"Socket client sent message: {message}");
                _hasDataToSend = false;
            }
            Thread.Sleep(0);
        }

        client.Shutdown(SocketShutdown.Both);
    }
}
```

Sending data from the client

```
internal class SocketClient
{
    private string _message = "";
    private bool _hasDataToSend = false;
    private bool _shallRun = true;

    public bool ShallRun { private get => _shallRun; set => _shallRun = value; }

    public void SendData(string message)
    {
        _message = message;
        _hasDataToSend = true;
    }

    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

…
```

Sending data from the client

```
        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream,
ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        while (ShallRun)
        {
            // Send message.
            if (_hasDataToSend)
            {
                var message = _message;
                var messageBytes = Encoding.UTF8.GetBytes(message);
                client.Send(messageBytes, SocketFlags.None);
                Console.WriteLine($"Socket client sent message: {message}");
                _hasDataToSend = false;
            }
            Thread.Sleep(0);
        }

        client.Shutdown(SocketShutdown.Both);
    }
}
```

Receiving data on the server

```
internal class SocketServer
{
    private bool _hasUnreadData = false;
    private string _data = "";
    public bool HasUnreadData { get => _hasUnreadData; private set => _hasUnreadData = value; }

    public string Data
    {
        get
        {
            HasUnreadData = false;
            return _data;
        }
        private set
        {
            _data = value;
            HasUnreadData = true;
        }
    }

    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            try
            {
                byte[] buffer = new byte[1024];
                int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
                string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
                Console.WriteLine($"Server received: {receivedData}");
                Data = receivedData;
            }
            catch (SocketException e)
            {
                Console.WriteLine("Got exception: " + e.ToString());
                break;
            }
        }
    }
}
```

Receiving data on the server

```
internal class SocketServer
{
    private bool _hasUnreadData = false;
    private string _data = "";
    public bool HasUnreadData { get => _hasUnreadData; private set => _hasUnreadData
= value; }

    public string Data
    {
        get
        {
            HasUnreadData = false;
            return _data;
        }
        private set
        {
            _data = value;
            HasUnreadData = true;
        }
    }
```

Receiving data on the server

```
public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp);

        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

…
```

Receiving data on the server

```
        var handler = listener.Accept();

        while (true)
        {
            try
            {
                byte[] buffer = new byte[1024];
                int numberOfBytesReceived = handler.Receive(buffer,
SocketFlags.None);
                string receivedData = Encoding.UTF8.GetString(buffer, 0,
numberOfBytesReceived);
                Console.WriteLine($"Server received: {receivedData}");
                Data = receivedData;
            }
            catch (SocketException e)
            {
                Console.WriteLine("Got exception: " + e.ToString());
                break;
            }
        }
    }
}
```


WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING

Problems in this approach

There are some problems with the way we do this:

- What if SendData() is called more than once, before the data is sent on the socket?
- What if SendData() is called before _hasDataToSend is reset?
- What if a user of the SocketServer reads the Data property at the same time as data is received?

Problems in this approach

There are some problems with the way we do this:

- What if SendData() is called more than once, before the data is sent on the socket?
- What if SendData() is called before _hasDataToSend is reset?
- What if a user of the SocketServer reads the Data property at the same time as data is received?

Some data might be lost!

Problems in this approach

There are some problems with the way we do this:

- What if SendData() is called more than once, before the data is sent on the socket?
- What if SendData() is called before _hasDataToSend is reset?
- What if a user of the SocketServer reads the Data property at the same time as data is received?

We will fix this in the coming weeks ☺

Your turn

Solve the

” Exercise - ECG Network – Without blocking collection”

References and image sources

Warning tape: https://www.pngall.com/warning-sign-png/download/69428

AARHUS UNIVERSITY