TCP Sockets and Web Server Basics

BY EVAN OLDS
CPT S 422

Sockets

- Sockets are network communication interfaces
  - Two computers connect via a socket connection
  - Send byte buffers
  - Receive byte buffers
  - That's about it with respect to the general concept

Sockets

- Rarely is the basic concept of what a socket is or what it does confusing
  - NetworkStream has Read and Write to receive and send buffers, respectively
- Hard parts are…
  - What they guarantee or don't guarantee during operations (i.e. "partial" transfers)
  - Associated threading/concurrency concepts

TcpClient and TcpListener

- Several types of sockets available, but we want to use TCP (Transmission Control Protocol)
  - Ensures that if we sent packets (chunks of data), X, Y and Z in that order, then they arrive as X, Y and Z in that order on the other end
  - If you want to know exactly how this is ensured, it's covered in networking classes
- .NET has TcpClient and TcpListener classes for TCP/IP (internet protocol) connections
- This is what HTTP (Hypertext Transfer Protocol) uses

Networking Basics

- TCP sockets connect two different computers together over the internet (or a local network)
- Need 2 things to construct the client socket
  - Port number (must match the port that the server is listening on)
  - Server IP address
- Server has a listener (TcpListener) waiting on the port, client connects, data is sent back and forth
- Again, a networking class covers these in details, we only need the high-level overview to build a web server

Using TcpListener

- To create a TcpListener and get it to listen for incoming connections:

```
TcpListener listener = new TcpListener(IPAddress.Any, port);
listener.Start();
TcpClient client = listener.AcceptTcpClient();
```

- Go to the MSDN documentation to understand these functions!
  - TcpListener constructor
  - TcpListener.Start
  - TcpListener.AcceptTcpClient

Web Server Pseudo-Code

```
while (active) {
    // blocking call to accept client
    client = someTcpListener.AcceptClient();

    ReadRequestFromClient();
    if (request_was_valid) {
        WriteResponseToClient();
    }
    client.Dispose();
}
```

Web Server Logic

- That's really all there is to it:
  - Accept a connection (on port 80 by default, we will use 4220 or some other port for testing)
  - Read data from client, terminate connection if it's not a valid web request
  - Write appropriate response if it is a valid web request
- Raises the question: What does a valid request look like?

Anatomy of a Web Request

METHOD URL HTTPVersion\r\n
Header1Name: Header1Value\r\n
Header2Name: Header2Value\r\n
… (more headers if desired, each ending with \r\n)
\r\n
RequestBody

Anatomy of a Web Request

- Example:

```
GET /index.html HTTP/1.1\r\n
User-Agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116
Safari/537.36\r\n
\r\n
```

- In this case we see that the body can be empty. It often is for simple requests.

Anatomy of an HTTP Response

HTTPVersion StatusCode ReasonPhrase\r\n
Header1Name: Header1Value\r\n
Header2Name: Header2Value\r\n
… (more headers if desired, each ending with \r\n)
\r\n
ResponseBody

Anatomy of an HTTP Response

- Example:

HTTP/1.1 200 OK\r\n

Content-Type: text/html\r\n

\r\n

<html>Hello World!</html>

HTTP Specification

- We will refer to it throughout the semester
- https://tools.ietf.org/html/rfc2616
- For now it's just a reference link, we'll start decoding more of its jargon later on
  - "CRLF" = "\r\n"
  - "SP" = " "
    - Space character
  - "LWS" = linear white space
  - "NLWS" = non-linear white space

Let's Make a Web Server

- Don't need to read 176 from the RFC pages to understand the basics
- Let's make a basic web server in class!