Module 6 – Node.js and Socket.IO

- Module 6 Contains 2 components
  - Individual Assignment and Group Assignment
    - Both are due on Wednesday November 15th

- Read the WIKI before starting

- Portions of today’s slides came from
  - Craig Hecock
  - Iván Loire
  - Roberto Peon
  - Charles Wang

Preview of Creative Project – Due 12/6/17

- Come up with an idea utilizing skills learned from previous modules
  - Should be comparable to work of previous modules
  - You may work alone or in a group on this module

- Create your Creative Project repo and push to it your project description along with a grading rubric after it is approved by a TA

- Rubric is due Wednesday November 15th by the end of class
  - The project is worth 100 points, 5 of which come from submitting this rubric to Bitbucket
    - You are allowed to assign up to 20 points for a creative portion
    - Create a file named gradingRubric.md inside your Bitbucket repo
    - Include the name of the TA that approved your rubric
CSE 503S Performance Evaluation Report

- 503S students will complete a performance evaluation study of their web server
- The evaluation will include two experiments that measure critical resources used throughout the semester
  - For example, if my app uses Apache and MySQL
    - How many pages/sec can Apache serve?
    - How many reads/sec and writes/sec can my database perform?
  - The wiki provides a list of potential experiments
- A performance evaluation proposal is due on November 15th (along with your Creative Project Description)
  - Create a Performance Evaluation Repo using the link provided on Piazza
  - List the experiments you plan to perform and why these are important to study
  - Create a file inside your Performance Evaluation Repo named `proposalExperiments.md` explaining your experiments by Wednesday
- Students will submit a written document explaining the experiments along with their results by 11:59 PM on Friday December 8th
- Refer to the course website for additional details about the report
  - https://classes.engineering.wustl.edu/cse330/index.php/CSE_503S_Performance_Evaluation_Study

CSE 503S Performance Evaluation Example
What is Node.js?

- A JavaScript runtime environment running Google Chrome’s V8 engine
  - a.k.a. a server-side solution for JS
  - Compiles JS, making it really fast

- Runs over the command line

- Designed for high concurrency
  - Without threads or new processes

- Never blocks, not even for I/O

Concurrency: The Event Loop

- Instead of threads Node uses an event loop with a stack

- Alleviates overhead of context switching
Non-blocking I/O

- Servers do nothing but I/O
  - Scripts waiting on I/O requests degrades performance

- To avoid blocking, Node makes use of the event driven nature of JS by attaching callbacks to I/O requests

- Scripts waiting on I/O waste no space because they get popped off the stack when their non-I/O related code finishes executing

I/O Example

```php
<?php
$result = mysql_query('SELECT * FROM ...');
while($r = mysql_fetch_array($result)){
  // Do something
}

// Wait for query processing to finish...
?>

<script type="text/javascript">
mysql.query('SELECT * FROM ...', function (err, result, fields){
  // Do something
});

// Don’t wait, just continue executing
</script>
Traditional (blocking) Threaded Model

- Blocking wastes cycles
- Waiting on File I/O
- Waiting on Network Response
- Waiting on Database

Node.js (non-blocking) Event Loop

- These are open for us to do more work!
- File I/O callback
- NET I/O callback
- DB I/O callback
Threads VS Event-driven

<table>
<thead>
<tr>
<th>Threads</th>
<th>Asynchronous Event-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock application / request with listener-workers threads</td>
<td>only one thread, which repeatedly fetches an event</td>
</tr>
<tr>
<td>Using incoming-request model</td>
<td>Using queue and then processes it</td>
</tr>
<tr>
<td>multithreaded server might block the request which might involve multiple events</td>
<td>manually saves state and then goes on to process the next event</td>
</tr>
<tr>
<td>Using context switching</td>
<td>no contention and no context switches</td>
</tr>
<tr>
<td>Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock</td>
<td>Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments</td>
</tr>
</tbody>
</table>

Node.js VS Apache

<table>
<thead>
<tr>
<th>Platform</th>
<th>Number of request per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP ( via Apache)</td>
<td>318,727</td>
</tr>
<tr>
<td>Static ( via Apache )</td>
<td>296,651</td>
</tr>
<tr>
<td>Node.js</td>
<td>556,930</td>
</tr>
</tbody>
</table>
Node.js Hello World (hello.js)

```javascript
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, world! ');
}).listen(80);
```

Node.js Examples
Module 6 Individual Demo

Socket.IO
Web Sockets

- Transport protocol for web communication
  - Alternative to the XMLHttpRequest object used for AJAX
- Provides Bi-directional, full duplex communication channel
- Designed for implementation in web browsers and servers
- Extremely low overhead for payload, just two bytes
- API standardized by W3C

HTTP Overhead (for each request)

“hello, my name is Chrome, encoding UTF-8... I would like a web page please.”

TCP handshake  +  HTTP Headers (request)  +  HTTP Headers (response)
• Page Load Time (PLT) is our measure of latency.

• Note the diminishing returns as bandwidth increases

---

Do closer server help?

• Now we vary the Round Trip Time (RTT) for a fixed bandwidth.

• Reducing RTT, always helps reduce Page Load Time (PLT).
Web Sockets

- Help reduce the number of Round Trip Times (RTTs)

**Browser**

**Server**

TCP handshake (just first request)

<table>
<thead>
<tr>
<th>Browser</th>
<th>Chrome</th>
<th>Safari</th>
<th>Opera</th>
<th>iOS Safari</th>
<th>Opera Mini</th>
<th>Android Browser</th>
<th>Chrome for Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE 8</td>
<td>45</td>
<td>46</td>
<td></td>
<td></td>
<td>8</td>
<td>4.3</td>
<td>4.4</td>
</tr>
<tr>
<td>IE 9</td>
<td>43</td>
<td>47</td>
<td></td>
<td></td>
<td>4.4</td>
<td>4.4</td>
<td></td>
</tr>
<tr>
<td>IE 10</td>
<td>44</td>
<td>48</td>
<td>9</td>
<td>8.4</td>
<td>4.4</td>
<td>4.4</td>
<td></td>
</tr>
<tr>
<td>IE 11</td>
<td>13</td>
<td>45</td>
<td>49</td>
<td>9.1</td>
<td>35</td>
<td>9.2</td>
<td>8</td>
</tr>
<tr>
<td>IE 14</td>
<td>46</td>
<td>50</td>
<td></td>
<td>36</td>
<td>9.3</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>IE 48</td>
<td>47</td>
<td>51</td>
<td></td>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**http://caniuse.com/#feat=websockets**

Web Sockets are a bidirectional communication technology for web apps. The table shows support across different browsers and operating systems.

- Reported to be supported in some Android 4+ browsers, including Sony Xperia S, Sony TX and HTC.
- Partial support refers to the websockets implementation using an older version of the protocol and/or the implementation being disabled by default (due to security issues with the older protocol).
- Partial support refers to lacking support for binary data.
Socket.IO

- **Web sockets for all**
  - Web sockets are not supported in all browsers

- **Socket IO supports a variety of transports**
  - HTML 5 WebSocket
  - Flash Socket
  - AJAX Long Polling
  - Forever Iframe
Getting Started with Socket.IO

• Socket.IO:  http://socket.io

• Install Socket.IO with npm:
  – npm install socket.io

Node example using Socket.IO:

```javascript
var app = require('http').createServer(callback);
var io = require('socket.io').listen(app);
app.listen(8080);
```

Socket.IO Server

```javascript
io.sockets.on('connection', function (socket) {
  socket.emit('Hello from server', { hello: 'world' });

  socket.on('Reply from client', function (data) {
    console.log(data);
  });
});
```
Socket.IO Client (index.html)

```html
<script src="/socket.io/socket.io.js"></script>

<script>
var socket = io.connect('http://localhost');
socket.on('Hello from server', function (data) {
    console.log(data);
    socket.emit('Reply from client', { hello: 'from client' });
});

</script>
```

Simple Socket.IO Demo
Simple Chat Server Demo