Node.js

Node.js is open source, high performance JavaScript server

- executing on top of Google's V8 engine
- originally written by Ryan Dahl in 2009

Node.js supports *high throughput*, *real-time*, *scalable* use

- via *asynchronous* and *event driven* API calls
- running as a *single non-blocking* thread
- *without buffering* data - it is output in chunks

However, Node.js isn't suitable for *CPU intensive* applications.

Node.js can be exercised on departmental Linux hosts thus:

```
linux% node
> console.log("Hello World!")
Hello World!
```

Its *Read Evaluate Print Loop* is similar to *jjs* REPL interface

```
linux% node
> n = 7
 7
> y = n - 1
 6
> car = { make: "Ford", model: "Focus", year: 2016 }
 { make: ’Ford’,
   model: ’Focus’,
   year: 2016 }
> car.model
 ’Focus’
> f = function() { for (var p in car) console.log(p) }
 [Function]
> f()
 make
 model
 year
 undefined
```

Expressions with no value get *undefined* reply from REPL.

Two Control Cs are required to stop REPL interpreter.

Node.js HTTP module

HTTP handler *hello.js* serves single web page on port 8080

```
var http = require("http");

var svr = http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World');
});

svr.on('error', function(err){ console.log('Server: ' + err); });

svr.listen(8080, function(){console.log('Node: linux02 port 8080');});
```

createServer() registers callback that's run by *request* event.

Code using http package can be run on *linux02* with

```
linux02% node hello.js
Node: linux02 port 8080
```

or loaded interactively with

```
linux02% node
> .load hello.js
Node: linux02 port 8080
```

It can be run (and killed) remotely on *linux02* via CGI program.

Inside HWU *hello.js* service is accessible at URL.

HWU firewall *fw1.hw.ac.uk* blocks access outside *hw.ac.uk* domain.

See this lecture's exercise for ways to bypass the HWU firewall.

Node Packages

Node modules are managed by the Node Package Manager *npm*.

```
unix% npm install express
```

installs package *express* in *./node_modules* for use with

```
var express = require("express");
```

Node.js Web Server

Node.js service *web.js* serves web pages using *fs* and *url* packages:

```
var fs = require('fs');
var http = require('http');
var url = require('url');

var svr = http.createServer( function (request, response) {
  var path = url.parse(request.url).pathname;
  var mimetype = "text/plain";
  var i = path.lastIndexOf(".");
  if ( i != -1 )
    switch ( path.substring(i) ) {
      case ".css":   mimetype = "text/css"; break;
      case ".gif":   mimetype = "image/gif"; break;
      case ".html":  mimetype = "text/html"; break;
      case ".jpg":   mimetype = "image/jpg"; break;
      case ".js":    mimetype = "text/javascript"; break;
      case ".png":   mimetype = "image/png"; break;
      case ".pdf":   mimetype = "application/pdf"; break;
      case ".svg":   mimetype = "image/svg+xml"; break;
      case ".xhtml": mimetype = "application/xhtml+xml"; break;
      case ".xml":   mimetype = "text/xml"; break;
      default:       i = -1;
    }

  fs.readFile("/home/hamish/www/wp" + path, function (err, data) {
    var code = 404;
    if ( err ) {
      mimetype = "text/plain";
      data = "file not found";
    } else if ( i == -1 ) {
      data = "invalid path";
    } else code = 200;
    response.writeHead( code, {'Content-Type': mimetype} );
    response.write( data );
    response.end();
  });
});

svr.on('error', function(err) {console.log('Server: ' + err);});

svr.listen(8081, function() {console.log('Node: linux03 port 8081');});
```

Web server

- asynchronously reads requested file without blocking
- uses callback to reply to HTTP request with file's contents

Run (or kill) node service on *linux03* via CGI. Access it online.

Using Express

Express is set of Node.js APIs that support web applications to

- set up middleware to respond to HTTP Requests
- define routing table to act based on HTTP request and URL
- dynamically render HTML pages via templates

app.js serves files and lists directory using fs module

```
var express = require('express');
var fs = require('fs');
var http = require('http');
var app = express();

var head = ["<!DOCTYPE html>", "<html>", "<head>",
            "<title>Images</title>", "</head>", "<body>" ];
var tail = ["</body>", "</html>"];
var directory = "/home/hamish/www/images";
var nl = "0;

// serve static files in images/ directory
app.use(express.static(directory));

// make virtual listing of images/ directory
app.get('/index.html', function (req, res) {
  fs.readdir(directory, function(err, files) {
    if ( err ) { return res.send(err); }
    var s = "";
    for (var i in head) s += head[i] + nl;
    s += "<h3>Directory: images/</h3>" + nl + "<ul>" + nl;
    files.forEach( function (file) {
      s += "<li><a href=
    });
    s += "</ul>" + nl;
    for (var i in tail) s += tail[i] + nl;
    res.send(s);
  });
});

var svr = http.createServer(app);
svr.on('error', function(err) {console.log('Server: ' + err);});
svr.listen(8080, function() {console.log("Node: linux04 port 8080");});
```

Web app can be run (and killed) remotely on *linux04*.

In HW access *app.js* at *http://linux04.macs.hw.ac.uk:8080/index.xhtml*.

Express Form Handling

Express can handle form data submitted via POST or GET methods.

```
<form action="http://linux10.macs.hw.ac.uk:8080/echo" method="post">
  <p> login <input type="text" name="login" required>
  <p> password <input type="password" name="pwd" required>
  <p> <input type="submit">
</form>
```

This form elicits a login and password from the user.

App still works if *method* attribute is changed to *get*.

Script *echo.js* echoes back names and values of submitted data

```
var express = require('express');
var http = require('http');
var app = express();
var bodyParser = require('body-parser');
var dataParser = bodyParser.urlencoded({extended: false});
var head = ["<!DOCTYPE html>", "<html>", "<head>",
            "<title>Form Data</title>", "</head>", "<body>" ];
var nl = '\n';

function handler(obj, res) {    // handles GET and POST data
  var s = "";
  for ( var i in head ) s += head[i] + nl;
  s += "<h3>Input Data</h3>" + nl;
  s += "<ul>" + nl;
  for ( var j in obj )
    s += "<li>" + j + " = " + obj[j] + "</li>" + nl;
  s += "</ul>" + nl + "</body>" + nl + "</html>" + nl;
  res.send(s);
}

app.get('/echo', function (req, res) { handler(req.query, res); });

app.post('/echo', dataParser, function (req, res) {
  handler(req.body, res);
});

var svr = http.createServer(app);
svr.on('error', function(err) { console.log('Server: ' + err); });
svr.listen(8080, function() {console.log("Node: linux10 port 8080");});
```

Web app can be run (and killed) remotely on *linux10*.

POST form data is parsed by Express's body-parser module.

Express Server Storage

Express can store/update shared *key/value* data in Node.js server

```
<form action='http://linux09.macs.hw.ac.uk:8081/store'>
  <p> Key <input name='key' value=''/> </p>
  <p> Value <input name='val' value=''/> </p>
  <p> <input type='submit'/> </p>
</form>
```

This form elicits a key and value from the user.

Script *store.js* updates and reports on stored data

```
var express = require('express');
var http = require('http');
var app = express();
var head = ["<!DOCTYPE html>", "<html>", "<head>",
            "<title>Store</title>", "</head>", "<body>"];
var url = "http://linux09.macs.hw.ac.uk:8081/store";
var tail = ["<form action='" + url + "'>",
            "<p>", "Key <input name='key' value=''>", "</p>",
            "<p>", "Value <input name='val' value=''>", "</p>",
            "<p>", "<input type='submit'/>", "</p>",
            "</form>", "</body>", "</html>"];
var store = new Object();
var eol = "\n";

app.get("/store", function (req, res) {
  if ( req.query.key != undefined && req.query.key != "" )
    if ( req.query.val != undefined && req.query.val != "" )
         store[req.query.key] = req.query.val;
    else delete store[req.query.key];
  var s = "";
  for (var i in head) s += head[i] + eol;
  s += "<p>" + eol;
  for (var i in store)
    s += i + " = " + store[i] + "<br/>" + eol;
  s += "</p>" + eol;
  for (var i in tail) s += tail[i] + eol;
  res.send(s); // send web page
});

var svr = http.createServer(app);
svr.on('error', function(err) { console.log('Server: ' + err); });
svr.listen(8081, function() {console.log("Node: linux09 port 8081");});
```

Web app can be run (and killed) remotely on *linux09*.

Server's sole thread avoids need to control concurrency of updates.

Node.js and File Databases

Web apps can support database operations over records held in a file.

Book catalogue file might contain lines of *title@author@year* data.

Node app *filedata.js* supports *retrieval* over such data

```
var express = require('express');
var http = require('http');
var fs = require('fs');
var app = express();
var storage = "/home/hamish/www/cgi-bin/na/storage.db";
var books = null;
var head = "<!DOCTYPE html>\n<html>\n";
head += "<head><title>Books</title></head>\n";

function book(t,a,y) { this.title = t; this.author = a;
                       this.year = y; }

function fileload() {
  books = new Array();
  fs.readFile(storage, (err, data) => {
    if (err) throw err;
    var list = data.toString('ascii').split("\n");
    for (var i=0; i<list.length-1; i++) {
      var b = list[i].split("@");
      books.push(new book(b[0], b[1], b[2]));
    }
  });
}

app.get('/search', function (req, res) {
  var qstring = req.query.term.trim();
  page = head + "<body>\n<table border='1'>\n"
  page += "<tr><th>title</th><th>author</th><th>year</th></tr>\n";
  for (var i=0; i<books.length; i++)
    if ( qstring == "*" ||
         (books[i].title == qstring || books[i].author == qstring) ) {
      page += "<tr><td>" + books[i].title + "</td>";
      page += "<td>" + books[i].author + "</td>";
      page += "<td>" + books[i].year + "</td></tr>\n";
    }
  res.end(page + "</table>\n</body>\n</html>\n");
});

var sv = http.createServer(app);
sv.on('error', function(err){ console.log('Node error: ' + err); });
sv.listen(8080, function(){ console.log("Node: linux08 port 8080"); });
fileload();
```

Web service can be run (and killed) remotely on *linux08*.

Node.js and File Databases

To support *creation* of books augment *filedata.js* with

```
function filesave() {
  var data = "";
  for (var i=0; i<books.length; i++)
    data += books[i].title + "@" + books[i].author + "@";
    data += books[i].year + "\n";
  fs.writeFile(storage, data, (err) => { if (err) throw err; });
}

app.get('/add', function (req, res) {
  var title = req.query.title.trim();
  var author = req.query.author.trim();
  var year = req.query.year.trim();
  books.push(new book(title, author, year));
  res.end(head + "<p>new record added</p>\n</body>\n</html>\n");
  filesave();
});
```

It adds new record to *books* and then saves the array in a file.

To support *deletion* of books augment *filedata.js* with

```
app.get('/delete', function (req, res) {
  var title = req.query.title.trim();
  var i = 0;
  while ( i < books.length && title != books[i].title )
    i++;
  if ( i < books.length ) {
    books.splice(i, 1);
    reply = "1 record deleted";
  } else reply = "record not found";
  res.end(head + "<p>" + reply + "</p>\n</body>\n</html>\n");
  if ( reply == "1 record deleted" )
    filesave();
});
```

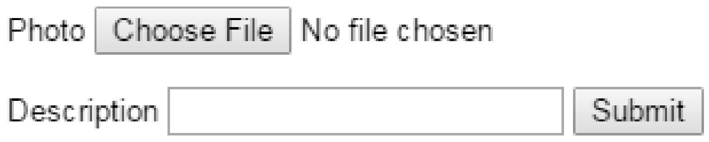
To support *updating* of years of books augment *filedata.js* with

```
app.get('/update', function (req, res) {
  var n = 0;
  for (var i=0; i<books.length; i++)
    if ( req.query.title == books[i].title ) {
      books[i].year = req.query.year;
      n++;
    }
  res.end(head + "<p>" + n + " updates</p>\n</body>\n</html>\n");
  if ( n != 0 )
    filesave();
});
```

Node.js File Upload

Node.js provides *multer* package to handle file uploading

```
<form action="http://linux05.macs.hw.ac.uk:8081/upload"
      method="post" enctype="multipart/form-data"> <p>
  Photo <input type="file" name="photo" required> <p>
  Description <input name="label" required> <input type="submit">
</form>
```



Multer parses *multipart/form-data* submissions as per RFC 1867

Upload can be handled by *upload.js*

```
var express = require('express');
var fs = require('fs');
var http = require('http');
var multer = require('multer');
var app = express();
var directory = '/tmp/upload/';
var upload = multer({dest: directory});
app.use(express.static(directory));

app.post('/upload', upload.single('photo'),
                    function (req, res, next) {
   var head = ["<!DOCTYPE html>", "<html>", "<head>",
                "<title>Upload</title>", "</head>", "<body>" ];
   var tail = ["</body>", "</html>"];
   var s = "";
   for (var i in head) s += head[i] + '\n';
   var f = req.file.originalname;
   s += "<p>photo = <a href=\"" + f + "\">" + f + "</a><br>\n";
   s += "label = " + req.body.label + "</p>\n";
   for (var i in tail) s += tail[i] + '\n';
   res.mimetype = "text/html";
   res.end( s );
   var newname = directory + req.file.originalname;
   next = fs.rename(req.file.path, newname, function(err) {
     if ( err ) console.log('ERROR: ' + err);
   });
})

var svr = http.createServer(app);
svr.on('error', function(err) {console.log('Server: ' + err);});
svr.listen(8081, function() {console.log("Node: linux05 port 8081");});
```

Web service can be run (and killed) remotely on *linux05*.

Pros and Cons of Node.js

Node.js is *innovative* web server technology that offers

- JavaScript for use as web server programming language
- *high throughput*, *real-time*, *scalable* web server apps

Drawbacks of Node.js for web server app development include

- coding using *callbacks* and *non-blocking IO* is harder
- flow control among callbacks and via event handling is trickier
- unsuitability for CPU intensive applications

Traditional web application servers like Tomcat

- support a wide range of web server applications
- host server applications from multiple different users
- execute multiple server applications via multithreading

In contrast Node.js servers

- only typically support a few applications each
- aren’t often used for hosting various apps from multiple users
- execute multiple server apps via a single service thread

Node.js requires users to write their JavaScript code to

- set up event handlers and register callbacks
- only use non-blocking asynchronous IO with files, buffers etc.
- decompose long program tasks into multiple, short-lived callbacks

It swaps a *callback hell* for prior *multi-threading hell*.

Things To Do

Read	Node.js Tutorial http://www.tutorialspoint.com/nodejs/
Browse	Node.js Website https://nodejs.org/
Try	*Quiz on Lecture*
Do	*Exercise 7*

Key Points

- ○ Node.js is a high performance platform based on Google's JavaScript V8 engine that supports an event-driven architecture and non-blocking I/O APIs to achieve high throughput and scalability for real-time web applications.

- ○ Node.js registers callbacks that are invoked when web service requests are received. The Express module is typically used to simplify routing and to handle requests to static pages.