Chapter 08
Node.js

Open Source SW Development
CSE22300

Thread VS Event

Thread

- **An abstraction of a unit of execution**
  - **Tasks**
- **What is in a task**
  - **Instructions to executes**
  - **Memory**
  - **File descriptors**
  - **Credentials**
  - **Locks**
  - **Network resources**

Thread

- **Threads**
  - **Share memory**
  - **Share file descriptors**
  - **Share filesystem context**

- **Processes**
  - **Not share memory**
  - **Not share most file descriptors**
  - **Not share filesystem context**

Thread Switching

- **To switch from thread T1 to T2:**
  - **Thread T1 saves its registers (including pc) on its stack**
  - **Scheduler remembers T1's stack pointer**
  - **Scheduler restores T2' stack pointer**
  - **T2 restores its registers**
  - **T2 resumes**

Thread Scheduler

- **Maintains the stack pointer of each thread**
- **Decides what thread to run next**
  - **E.g., based on priority or resource usage**
- **Decides when to pre-empt a running thread**
  - **E.g., based on a timer**
- **Needs to deal with multiple cores**
  - **Didn’t use to be the case**
- **“fork” creates a new thread**

Uses of Threads

- **To exploit CPU parallelism**
  - **Run two CPUs at once in the same program**
- **To exploit I/O parallelism**
  - **Run I/O while computing, or do multiple I/O**
  - **I/O may be “remote procedure call”**
- **Effective CPU intensive work**

Common Problems

- **Priority Inversion**
  - **High priority thread waits for low priority thread**
  - **Solution: temporarily push priority up (rejected??)**
- **Deadlock**
  - **X waits for Y, Y waits for X**
- **Incorrect Synchronization**
  - **Forgetting to release a lock**
- **Failed “fork”**
- **Tuning**
  - **E.g. timer values in different environment**

Event

- **An object queued for some module**
- **Operations:**
  - **create_event_queue(handler) → EQ**
  - **enqueue_event(EQ, event-object)**
  - **Invokes, eventually, handler(event-object)**
- **Handler is not allowed to block**
  - **Blocking could cause entire system to block**
  - **But page faults, garbage collection,**

Event Scheduler

- **Decides which event queue to handle next.**
  - **Based on priority, CPU usage, etc.**
- **Never pre-empts event handlers!**
  - **No need for stack / event handler**
- **May need to deal with multiple CPUs**

Event Synchronization

- **Handlers cannot block → no synchronization**
- **Handlers should not share memory**
  - **At least not in parallel**
- **All communication through events**

Uses of Events

- **CPU parallelism**
  - **Different handlers on different CPUs**
- **I/O concurrency**
  - **Completion of I/O signaled by event**
  - **Other activities can happen in parallel**

Threads vs. Events

- **Events-based systems use fewer resources**
  - **Better performance (particularly scalability)**
- **Event-based systems harder to program**
  - **Have to avoid blocking at all cost**
  - **Block-structured programming doesn't work**
  - **How to do exception handling?**
- **In both cases, tuning is difficult**

Node.js

NodeJS

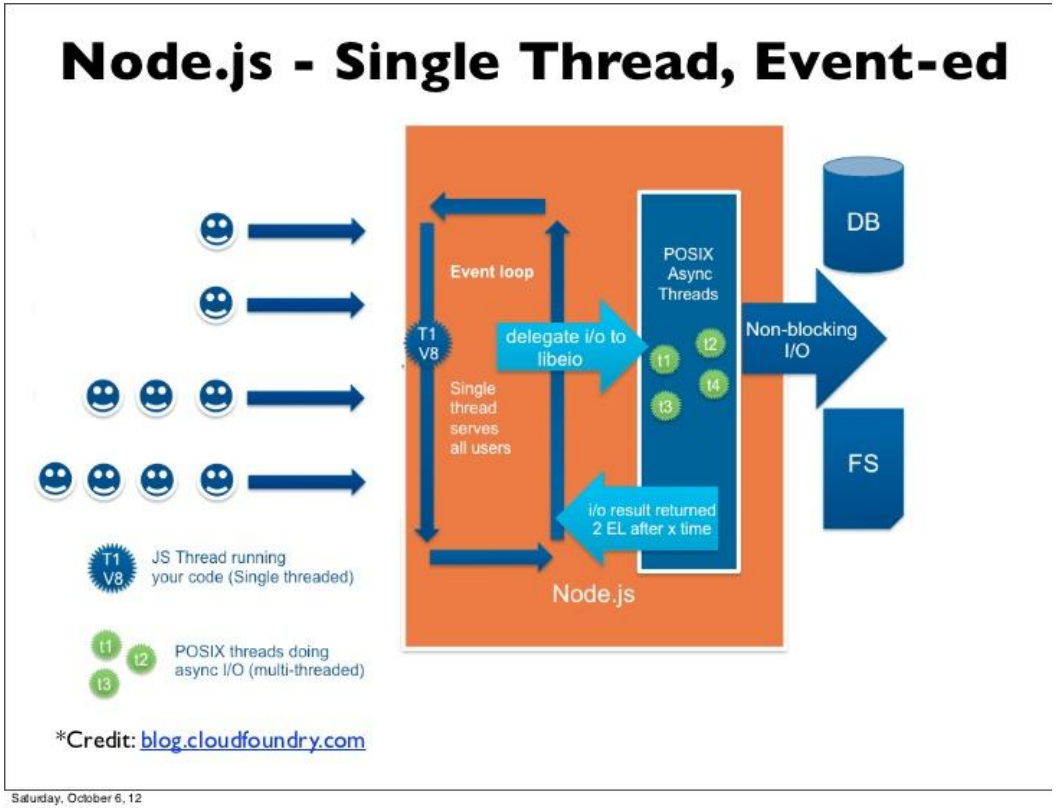
- **Uses Googles V8 JavaScript engine as an interpreter**
  - **Very fast**
- **Not a JavaScript framework**
  - **A Runtime Environment**
- **Event-driven architecture**
  - **All APIs of Node.js library are asynchronous**
- **Non-blocking I/O**
- **Single thread**

NodeJS

- **Contains many modules**
  - **Similar to the built-in java object**
- **Loading modules simply uses ‘require(..)’**
- **All built in (and modules loaded from npm) do not need pathing**
- **Modules created in application need path**

Advantages : Asynchronous

- **No waiting time for CPU**
  - **No blocking**
  - **Asynchronous I/O**



Advantages: Productivity

- **Tearing down barrier for Web backend**
  - **Web backend needs lots of background knowledge for developer**
  - **JavaScript is used for Web front-end**
  - **With Node.js, Web front-end developer can code backend-apps**
  - **Easy access for Web backend**
- **No Threads**
  - **No Synchronization, Critical section**
  - **Simple programming**
- **Modules**
  - **NPM**

Disadvantages: Single Thread

- **Intensive CPU Job**
  - **A certain job can be bottleneck to Node.js**
  - **I/O intensive job is sweet spot for Node.js**

- **Multiple Core**
  - **Node.js can be run on single thread and Core**
  - **Other cores??**
  - **Cluster Module**
  - **Session Sharing??**

Disadvantages: Call Back

- **Readability**
  - **JavaScript vs. Java**

- **Call Back Hell**

node95.js

```
var floppy = require('floppy');

floppy.load('disk1', function (data1) {
    floppy.prompt('Please insert disk 2', function () {
        floppy.load('disk2', function (data2) {
            floppy.prompt('Please insert disk 3', function () {
                floppy.load('disk3', function (data3) {
                    floppy.prompt('Please insert disk 4', function () {
                        floppy.load('disk4', function (data4) {
                            floppy.prompt('Please insert disk 5', function () {
                                floppy.load('disk5', function (data5) {
                                    // if node.js would have existed in 1995
                                });
                            });
                        });
                    });
                });
            });
        });
    });
});

```

Disadvantages: Learning Curve

- **Sequential Programming → Event Programming**

```
FILE *pFile = fopen("file.txt", "r");  // Wait...
char buf[101];
if ( NULL != pFile ) {
  fread(buf, 1, 100, pFile); // Wait...
  buf[(sizeof buf)-1] = 0;
  printf("%s", buf); // Wait...
  fclose(pFile); // Wait...
}
printf("Exit");
```

```
var file = "somefile.txt";
var fs = require("fs");
rs.readFile(file, function(err, data) {
  if ( err ) { throw err; }
  var len = data.toString().split("\n").length -1;   console.log("File Length: " + len);
});
console.log("Exit");
```

NPM

Node Package Manager

- **Online repositories for node.js**
- **Command line utility to install node.js package**
- **NPM comes bundled with node.js installables**

- **Search Modules**
  - **npm search <search name>**
- **Installing Modules**
  - **npm install <module name>**
- **Uninstalling Modules**
  - **npm uninstall <module name>**

- **How to Use**
  - **var express = require('express');**

Global vs Local

- **Default/Local**
  - **NPM installs any dependency in the local mode**
  - **Installed Packages/dependencies in node_modules directory**
  - **Deployed packages are accessible via require() method**

- **Global**
  - **Installed packaged/dependencies in system directory**
  - **Directly run installed packages**
  - **Cannot import using require()**

Package.json

- **Define the properties of a package**
  - **Package.json is present in the root directory of any Node application/module**

- **Attributes of Packages.json**
  - **name : name of the package**
  - **version : version of the package**
  - **description : description of the package**
  - **homepage : homepage of the package**
  - **author : author of the package**
  - **contributors : name of the contributors to the package**
  - **dependencies : list of dependencies**
  - **repository : repository type and URL of the package**
  - **main : entry point of the package**
  - **keywords : keywords**

Package.json

- **Create your own packages.json**
  - **npm init**
- **Add dependencies in your packages.json**
  - **npm install <package> --save**

- **Exersize**
  - **Create a folder**
  - **npm init**
  - **npm install express –save**
  - **Check packages.json**

Your Own Module

- **Use exports keyword**
  - Make properties and methods available outside the module file

- myfirstmodule.js

```
exports.myDateTime = function () {
    return Date();
};
```

- app.js

```
var dt = require('./myfirstmodule');

console.log(dt.myDateTime());
```

Callback Concept

Blocking Code

- **Callback is an asynchronous function**
  - **Callback is called at the completion of a given task**

- **Blocking Code Example**
  - **Creates input.txt**
  - **npm install fs –save**
  - **Copy codes as block.js**

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');
console.log(data.toString());
console.log("Program Ended");
```

  - **node block.js**

Non-Blocking Code

- **Non-Blocking Code Example**
  - **Creates input.txt**
  - **npm install fs –save**
  - **Copy codes as nonblock.js**

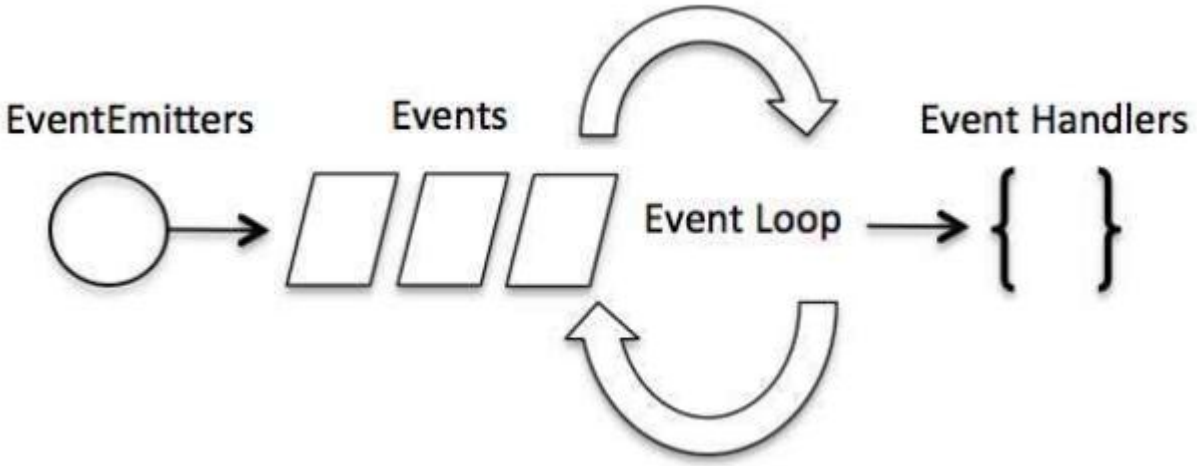
```
var fs = require("fs");
fs.readFile('input.txt', function (err, data) {
    if (err) return console.error(err);
        console.log(data.toString());
});
console.log("Program Ended");
```

  - **node nonblock.js**

Event Loop

Event-Driven Programming

- **Node.js is a single-threaded application**
  - **Event and Callbacks**
  - **Triggers a callback function when one of those events is detected**



Event-Driven Programming

- **Import events module**
  - **var events = require(‘events’);**
- **Create an eventEmitter object**
  - **var eventEmitter = new events.EventEmitter();**
- **Bind event and event handler**
  - **eventEmitter.on(‘eventName’, eventHandler);**
- **Fire an event**
  - **eventEmitter.emit(‘eventName’);**

Event-Driven Programming

- **Example**

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
var connectHandler = function connected() {
    console.log('connection successful.');
    eventEmitter.emit('data_received');
}

eventEmitter.on('connection', connectHandler);

eventEmitter.on('data_received', function(){
   console.log('data received successfully.');
});

eventEmitter.emit('connection');
console.log("Program Ended.");
```

Event Emitter

Method	Description
addListener(event, listener) on(event, listener)	Adds a listener at the end of the listeners array for the specified event.
once(event, listener)	Adds a one-time listener to the event. This listener is invoked only the next time the event is fired, after which it is removed.
removeListener(event, listener)	Removes a listener from the listener array for the specified event.
removeAllListeners([event])	Removes all listeners, or those of the specified event.
setMaxListeners(n)	By default, EventEmitters will print a warning if more than 10 listeners are added for a particular event.

Event Emitter

Method	Description
listeners(event)	Returns an array of listeners for the specified event.
emit(event, [arg1], [arg2], [...])	Execute each of the listeners in order with the supplied arguments. Returns true if the event had listeners, false otherwise.

Event Emmitter

- **Example**

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
var listner1 = function listner1() {
console.log('listner1 executed.');
}
var listner2 = function listner2() {
console.log('listner2 executed.');
}

eventEmitter.addListener('connection', listner1);
eventEmitter.on('connection', listner2);
var eventListeners = require('events').EventEmitter.listenerCount(eventEmitter,'connection');
console.log(eventListeners + " Listner(s) listening to connection event");

eventEmitter.emit('connection');
eventEmitter.removeListener('connection', listner1);
console.log("Listner1 will not listen now.");

eventEmitter.emit('connection');
eventListeners = require('events').EventEmitter.listenerCount(eventEmitter,'connection');
console.log(eventListeners + " Listner(s) listening to connection event");
console.log("Program Ended.");
```

File System

Open File

- **Standard POSIX functions**
- **Import FS**
  - **var fs = require(“fs”);**

- **Open a File**
  - **fs.open(path, flags, [mode], callback);**
  - **path : file name path**
  - **flags : read/write flags**
  - **mode : file permission only if the file was created**
  - **callback : get two arguments (err, fd)**

Open File

- **Example**

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

console.log("Going to open file!");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
    console.log("File opened successfully!");
});
```

Writing File

- **Writing a File**
  - **fs.writeFile(filename, data,[options], callback)**
  - **path : file name path**
  - **data : String or buffer**
  - **mode : The third parameter is an object which will hold {encoding, mode, flag}**
  - **callback : get single argument (err)**

Writing File

- **Example**

```
var fs = require("fs");
console.log("Going to write into existing file");
fs.writeFile('input.txt', 'Simply Easy Learning!', function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("Data written successfully!");
   console.log("Let's read newly written data");
   fs.readFile('input.txt', function (err, data) {
      if (err) {
         return console.error(err);
      }
      console.log("Asynchronous read: " + data.toString());
    });
})
```

Reading File

- **Reading a File**
  - **fs.read(fd, buffer, offset, length, position, callback)**
  - **path : file name path**
  - **buffer : The data will be written to**
  - **offset : the buffer to start writing at**
  - **length : The number of bytes to read**
  - **position : where to begin reading from in the file**
  - **callback : get single arguments (err)**

Reading File

- **Example**

```
var fs = require("fs");
var buf = new Buffer(1024);
console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to read the file");
   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
        if (err){
             console.log(err);
        }
        console.log(bytes + " bytes read");
        if(bytes > 0){
             console.log(buf.slice(0, bytes).toString());
        }
    });
});
```

Deleting File

- **Deleting a File**
  - **fs.unlink(path, callback)**
  - **path : file name path**
  - **callback : get single argument (err)**

Deleting File

- **Example**

```
var fs = require("fs");
console.log("Going to delete an existing file");
fs.unlink('input.txt', function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("File deleted successfully!");
});
```

Creating Directory

- **Creating a directory**
  - **fs.mkdir(path[, mode], callback)**
  - **path : file name path**
  - **mode : directory permission to be set**
  - **callback : get single argument (err)**

Creating Directory

- **Example**

```
var fs = require("fs");
console.log("Going to create directory /tmp/test");

fs.mkdir('/tmp/test',function(err){
   if (err) {
       return console.error(err);
   }
   console.log("Directory created successfully!");
});
```

Others

- **Closing a File**
  - **fs.close(fd, callback)**
- **Truncate a File**
  - **fs.ftruncate(fd, len, callback)**
- **Read a Directory**
  - **fs.readdir(path, callback)**
- **Remove a Directory**
  - **fs.rmdir(path, callback)**