**Node advance course:**

**Node != JavaScript**

Node’s Architecture: V8 and libuv:

Node’s default VM is V8 to execute JS code. One other option is Chakra engine by Microsoft, it powers the Microsoft edge browser.

A screenshot of a social media post

Description automatically generated

Shipping features are on by default. Staged and In Progress features are not, we can use command line flags to enable them.

We can use --harmony flag to enable staged.

In progress features are less stable but we can still enable them if we want with specific flags.

A screenshot of a cell phone

Description automatically generated

Node uses V8 via V8’s C++ Bindings APIs.

Node also handles the waiting for asynchronous events for us using **libuv**

**There is only one thread in node, so there is no locking or race condition involved. At a time only one callback can be executed. At the time of callback node passes controls to V8 then after done with execution V8 passes control to node.**

Node’s CLI and REPL:

For starting REPL – type “node” in CLI and hit enter.

Global Objects, Process, and Buffer Objects:

Node process object provides a bridge between node application and its running environment.

Process object, commands

$ node -p “process.versions”

Output

{

node: '12.13.1',

v8: '7.7.299.13-node.16',

uv: '1.33.1',

zlib: '1.2.11',

brotli: '1.0.7',

ares: '1.15.0',

modules: '72',

nghttp2: '1.39.2',

napi: '5',

llhttp: '1.1.4',

http\_parser: '2.8.0',

openssl: '1.1.1d',

cldr: '35.1',

icu: '64.2',

tz: '2019c',

unicode: '12.1'

}

$ node -p “process.env” | less

The env property exposes a copy of the user environment. The out will be same as “env” command in linux.

We usually read password or API keys from environment variables. Ports details, which database URI to connect to, all is read from Environment variables.

$ node -p "process.release.lts"

If this node release is LTS then it will give output otherwise it will be undefined.

// Process is an event emitter

Buffer class is also available on global object and is used heavily in node to work with binary streams of data. A buffer is essentially a chunk of memory allocated out side V8. A buffer is a lower level data structure to represent the sequence of binary data and unlike arrays once a buffer is allocated it cannot be resized.

Allocating buffer –

$ Buffer.alloc(8) – It will automatically assign the value

$ Buffer.allocUnsafe(8) – It does not assign the value

$ Buffer.allocUnsafe(8).fill() – It will fill the data

How require() Actually Works:

Require() is available on global object.

A screenshot of a cell phone

Description automatically generated

Resolving – Define the absolute file path of a module.

Loading – It is determined by the content of the file at the result path.

Wrapping – It gives every module its private scope and what makes require local to every module.

Evaluating – VM eventually does with the code

Caching – If we will require this module again then we don’t have to go to all the steps again.

require(‘find-me’) – Node will start finding the module from current directory and it will go all the way up to root directory and if it still does not finds the module it will throw an error ( can not find module ).

To be exact node will look for ‘find-me.js’ in more folders –

$HOME/.node\_modules

$HOME/.node\_libraries

$PREFIX/lib/node

Core node modules are exception here. The result steps return immediatelty.

Like – const fs = require(‘fs’);

How node resolves non core modules –

require.resolve(‘find-me’) – It will do all the steps of require but it will not load the file. This is used to check whether a package is installed or not.

$ require.extensions

{ ‘.js’: [Function], .json: [Function], ‘.node’: [Function] }

Wrapping and Caching Modules:

exports.id = 1; // this is ok

exports = { id: 1 }; // this is not ok ( we can not replace export object directly )

module.exports = { id: 1 }; // this is ok

Why? Why exports object can not be replaced directly?

var g = 42; // local to this file.

Answer is before compiling a module node wraps the module code in a function which we are going inspect using the wrapper property.

For seeing make any error in the file.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

There is nothing special about require. It’s a function that takes the module name and path and returns the exports object.

We can simply override the require function to do our own logic if we want to.

A screenshot of a computer screen

Description automatically generated

Caching:

If we require a file twice but the content will only be shown once because of caching.

A close up of text on a black background

Description automatically generated

Know your NPM:

npm i –dry-run // it will only report what will be installed

To see all the globally installed packeges

$ npm ls -g –depth=0

For Json format

$ npm ll -g --depth=0 –json

If you are installing a dependency you should document it in package.json.

If you are using

$ npm i -S – Then it is considered as production dependency

$ npm i -D – The dependency is considered as development dependency

$ npm I -O – Optional dependency

$ npm update command – To update either a single package or all installed packages when the package name is not provided.

Operators for version –

< or <= -

2.2.\* - can be used to cover the whole range for that level

~2.2.9 – 1.2.x only for all x > 9

^2.1.3 –

**Concurrency Model and Event Loop:**

What is I/O anyway:

I/O is used to label a communication between a process in a computer CPU and anything external to that CPU.

The Event Loop:

The entity that handles external events and converts them into callback invocations.

The Call Stack:

It is simply a list of functions. JavaScript is single threaded that means there is only 1 stack and it can do 1 thing at a time. If the stack is executing something nothing else will happen in that single thread. Exactly stack waala concept.

Handling Slow Operations:

In dealing slow operation the fact that we are dealing with single thread becomes a problem.

**Simulation example:**

We can still write blocking code in node. A long for loop

A screenshot of a cell phone

Description automatically generated

Node’s event loop exists to allow us to avoid doing this style of programming.

How Callbacks Actually Work:

See the video ( it is explained in best way in that video ).

Module 3 – Video 6

setImmediate and process.nextTick:

Always choose setImmediate in place of setTimeOut with 0 s waiting time.

Process.nextTick is used to make callbacks asynchronous.

**Node’s Event Driven:**

Node’s Common Built-in Modules:

Event Emitter:

It is a module that facilitates communication between objects in node. Event emitter is at the core of node async event driven architecture.

Arguments, Errors, and Order:

Practical Example: Task List Manager:

Practical example using node’s event emitter

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

where

help – to display the list of commands

ls – list the current tasks

add – to add a new task

delete – to delete the task

**Node For Networking:**

DNS Module:

UDP

**Node For Web:**

The Basic Streaming HTTP Server:

**Requesting HTTP/HTTPS Data:**

Parsing URLs and Query Strings:

**Node’s Common Built in Libraries:**

Working with Operating System:

Working with the File System:

All the fs module functions have asynchronous and synchronous forms. You can pick either form depending on your code logic. For example reading a file during server initialization process readFileSync is probably okay. But if you are reading a file every time a user request something from that server you should probably take the asynchronous form.

A screenshot of a computer

Description automatically generated

Async and Sync functions handle exception differently . The async functions pass any encountered error normally as first argument of callback while the synchronous function immediately throw any errors. When using the synchronous function if you don’t want the errors to stop program you need to use try catch block to handle them.

By default the type of data in readFile is buffer.

Console and Utilities:

**Working with Streams**

Stream all the things:

“Streams are Node’s best and most misunderstood idea.”

**What are streams:**

Collections of data that might not to be available at once and don’t have to fit in memory.

Data that is coming from external source one chunk at a time.

Steams 101:

4 fundamental types of Stream:

* Readable – is an abstraction for source from which data can be consumed.

fs.createReadStream method

* Writable – is an abstraction for the destination to which data can be given.

fs.createWriteStream method

* Duplex – both readable and writable. Net.socket
* Transform – basically duplex streams that can be used to modify or transform the data as it is written in it. zlib.createGzip

All streams are instances of event emitter. They all emit event that we can use to write or read data from them.

src.pipe(dst)

src – readable stream

dst – destination stream

A screenshot of a cell phone

Description automatically generated