

Is Node.js Really Single-Threaded?

A tutorial on multi-threading, multi-processing, threads, processes, thread pools, and more



Salil Arora

[Follow](#)

Sep 24, 2019 · 5 min read



If you're new to Node.js and you want to understand how Node.js works under the hood, then read this article.

Every Node.js article introduces Node.js as single-threaded. Below are some misconceptions about multi-threading in Node.js.

1. Just like JavaScript, Node.js doesn't support multi-threading.
2. Node.js is a proper multi-threaded language just like Java.
3. There are two threads in Node.js, one thread is dedicatedly responsible for the event loop and the other is for the execution of your program.

Well, relax, these are just assumptions that can be actually termed as misconceptions.

Most of us, at some point in our life, got confused between multi-threading and multi-processing, threads and processes, thread pool, and, finally, how operating systems manage to do so.

Let's clear the basics.

Process

A process is a program under execution, i.e. a running program and is created when a program starts execution. A process can have multiple threads.

Threads

A *thread* is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.

The primary difference is that threads within the same process run on a shared memory space, while processes run in separate memory spaces.

Multi-processing

Multiprocessing is the use of two or more CPUs (processors) within a single computer system. Now, as there are multiple processors available, multiple processes can be executed at a time.

Multi-threading

Multi-threading is an execution model that allows a single process to have multiple code blocks (threads) running concurrently within the “context” of that process.

Thread pool

A thread pool is a group of pre-instantiated, idle threads which stand ready to be given work. By maintaining a pool of threads, the model increases performance and avoids latency in execution, due to frequent creation and destruction of threads for short-lived tasks.

More detailed articles need to be written to explain the above concepts in depth but let's not deviate from our agenda.

A Node.js application runs on single thread and the event loop also runs on the same thread. Hence, we can say Node.js is single-threaded but the catch is that there are some libraries in Node.js that are not single-threaded.

Let's understand this with the help of an example.

```
1  const crypto = require("crypto");
2  const start = Date.now();
3
4  function logHashTime() {
5      crypto.pbkdf2("a", "b", 100000, 512, "sha512", () => {
6          console.log("Hash: ", Date.now() - start);
7      });
8  }
9  logHashTime();
10 logHashTime();
11 logHashTime();
```

Here, I am importing Node's `crypto` module and I am calling its `pbkdf2` function four times and displaying the time taken inside the returned callback.

Curious to see the output? Let's execute this.

```
→ server1 git:(master) ✗ node threads.js
Hash: 1213
Hash: 1216
Hash: 1222
Hash: 1224
```

If you see the output, the four `logHashTime()` functions took almost the same time to execute. Let's understand why they are all taking a similar time to execute.

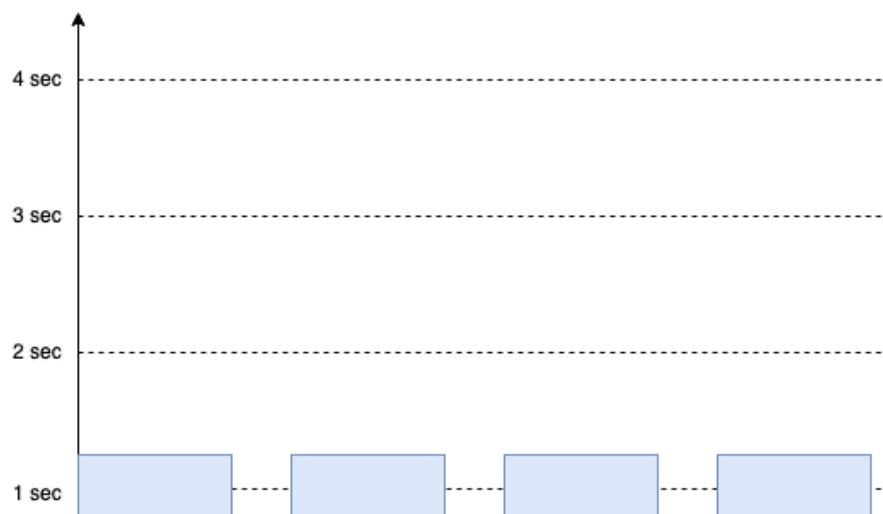
Node.js internally uses the `libuv` library which is responsible for handling operating system related tasks, like asynchronous I/O based operation systems, networking, concurrency, etc.

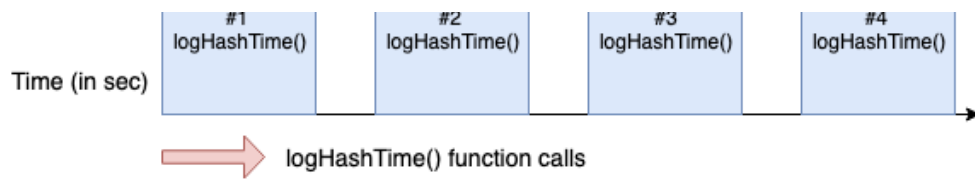
If you want to understand the workings of `libuv` in detail then refer to this [article](#).

`Libuv` sets up a thread pool of four threads to perform OS-related operations by utilizing the power of all the CPU cores. Given our machine has four cores, each thread from the pool is assigned to every core.

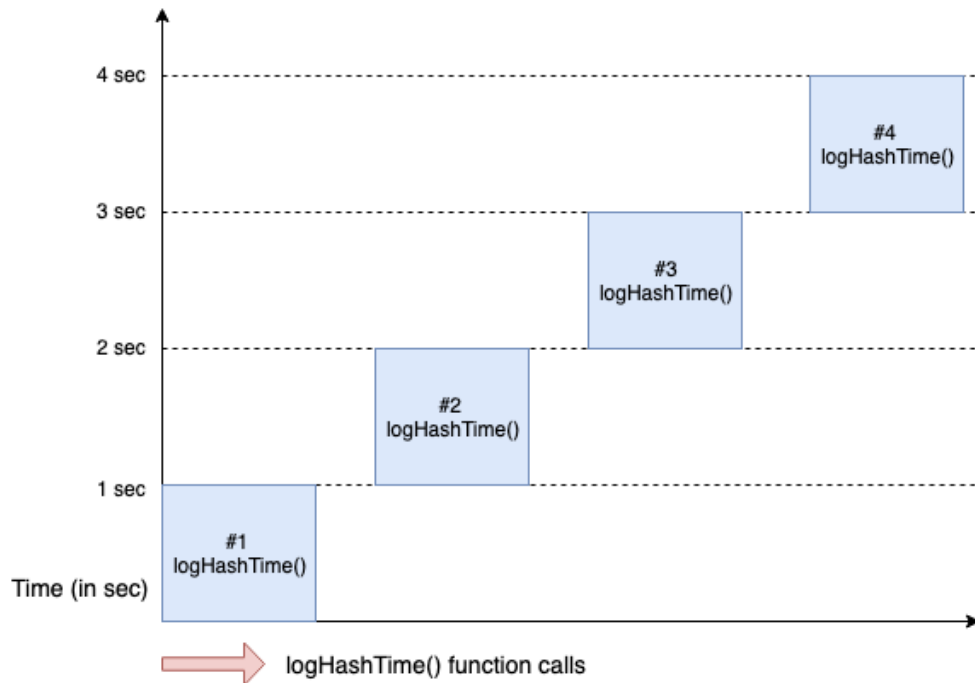
This results in one thread per core. With this setup, all the four threads will execute `logHashTime()` in each core in parallel, which makes all the four functions take a similar amount of time.

Let's visualize the execution of `logHashTime()` :

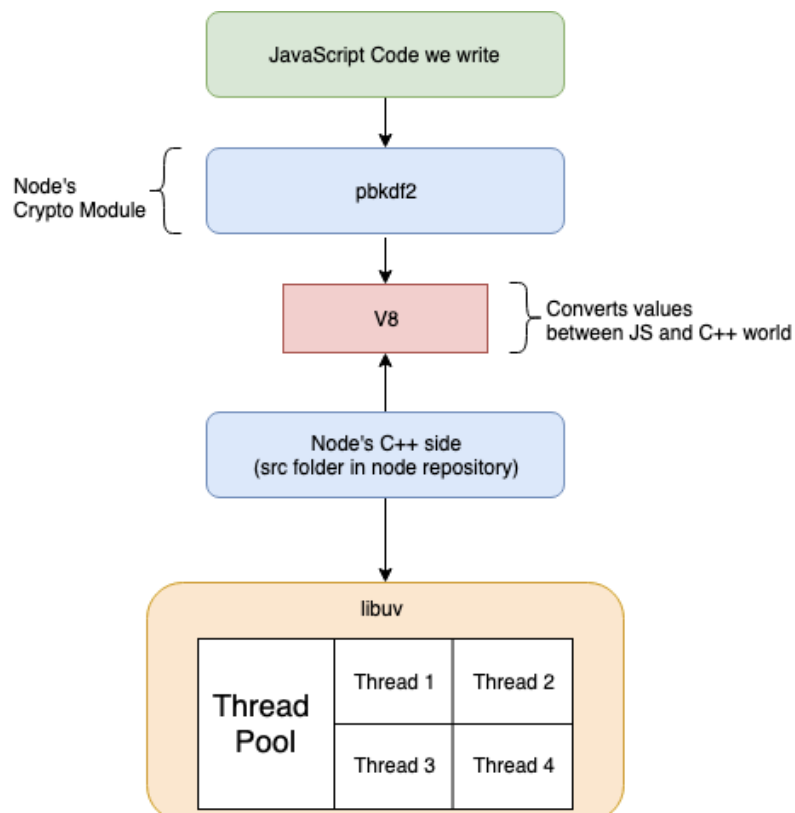




If this was not Node.js, this is how it should have been executed.



Let's see a high-level view of how all of these work together:



With the above concept, a lot of what-ifs would definitely arise.

• • •

What If I Have Two Cores and Multiple Operations to Perform?

Currently, my machine has four cores, and let's modify the previous example and call `logHashTime()` five times.

```
1  const crypto = require("crypto");
2  const start = Date.now();
3
4  function logHashTime() {
5    crypto.pbkdf2("a", "b", 100000, 512, "sha512", () => {
6      console.log("Hash: ", Date.now() - start);
7    });
8  }
9  logHashTime();
10 logHashTime();
11 logHashTime();
12 logHashTime();
13 logHashTime();
```

fiveThreads.js hosted with ❤ by GitHub

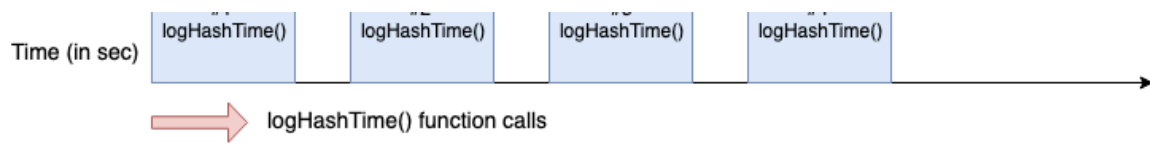
[view raw](#)

Let's check the output.

```
→ server1 git:(master) x node threads.js
Hash: 1222
Hash: 1258
Hash: 1265
Hash: 1310
Hash: 1833
```

Please notice that there is a difference between the first four calls (1.2 seconds average) vs. the fifth call (1.8 seconds).

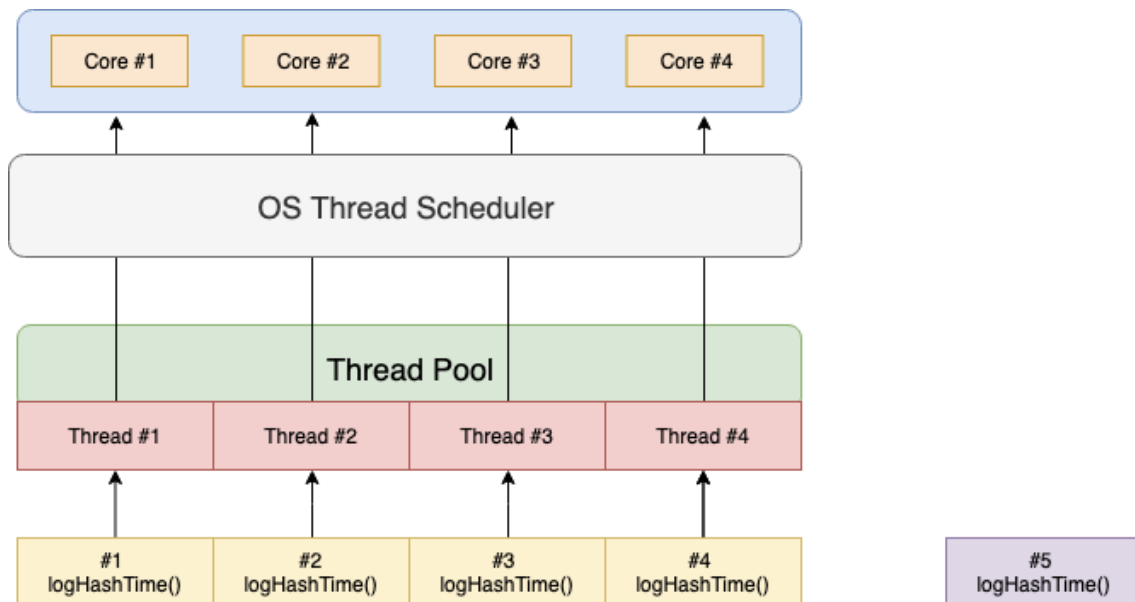




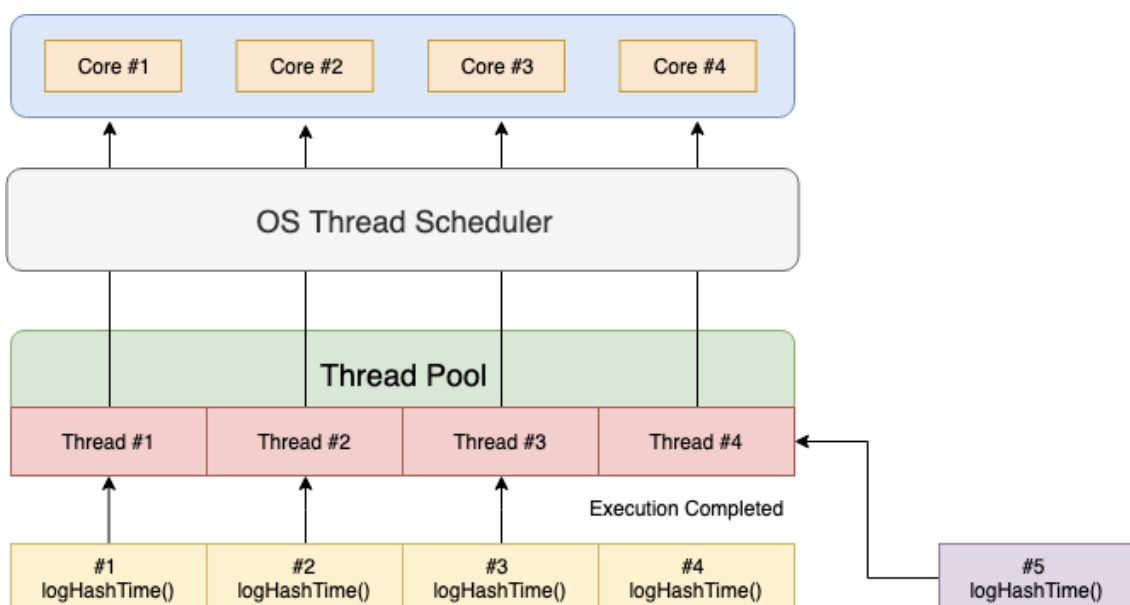
Libuv creates a thread pool of four threads that will execute `logHashTime()` functions on four cores in a timeframe of one second each.

When any thread completes execution, it will pick the fifth `logHashTime()` function to execute. So, that's why we can see a clear time difference between the first four calls and the fifth call.

Let's visualize this.



In our case, the fourth thread finishes execution and is ready to pick the fifth `logHashTime()` function.



. . .

What If I Want to Tweak the Number of Threads in the Libuv Thread Pool?

We can tweak the number of threads in libuv's thread pool by writing a single line of code.

```
process.env.UV_THREADPOOL_SIZE = noOfThreads;
```

Let's take the same example and increase the number of threads from the default four to five.

```
1 process.env.UV_THREADPOOL_SIZE=5;
2 const crypto = require("crypto");
3 const start = Date.now();
4
5 function logHashTime() {
6   crypto.pbkdf2("a", "b", 100000, 512, "sha512", () => {
7     console.log("Hash: ", Date.now() - start);
8   });
9 }
10 logHashTime();
11 logHashTime();
12 logHashTime();
13 logHashTime();
14 logHashTime();
```

tweakthreads.js hosted with ❤ by GitHub

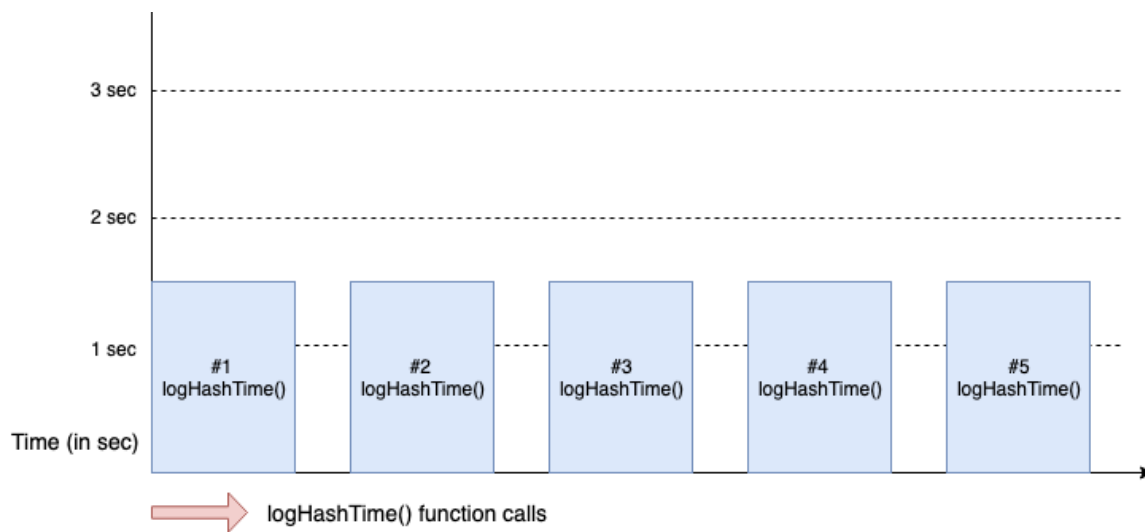
[view raw](#)

Let's check the output.

```
→ server1 git:(master) ✗ node threads.js
Hash: 1545
Hash: 1549
Hash: 1554
Hash: 1555
Hash: 1557
```

By tweaking the number of threads, we can clearly see that all five `logHashTime()` functions are taking an equivalent time of 1.5 seconds.





Why is this happening?

All five threads are trying to execute the `logHashTime()` function on the four CPU cores. So, the OS thread scheduler balances and gives equal time to each thread to finish execution with the help of context switching.

The third what-if will be covered in the next article.

. . .

Conclusion

That's how we can conclude Node.js is single-threaded but in the background it uses *multiple threads* to execute asynchronous code with the help of the libuv library.

. . .

What's Next?


There are many more topics we need to discuss in detail. In the next few posts, I will write in detail about the following topics:


- How the Node event loop works, internally.
- Enhancing Node.js performance through clustering.
- Hapi or Express for your startup.

Thanks for reading.

Thanks to Mahesh Haldar.

Get the Medium app

 A button that says 'Download on the App Store', and if clicked it will lead you to the iOS App store

 A button that says 'Get it on, Google Play', and if clicked it will lead you to the Google Play store