**Node.js Multithreading: Executing Callbacks in Separate Threads**

How to perform multi-threading on Node.js

[[nairihar](https://nairihar.medium.com/?source=post_page-----39e83a0a9ded--------------------------------)](https://nairihar.medium.com/?source=post_page-----39e83a0a9ded--------------------------------)

[[Bits and Pieces](https://blog.bitsrc.io/?source=post_page-----39e83a0a9ded--------------------------------)](https://blog.bitsrc.io/?source=post_page-----39e83a0a9ded--------------------------------)

[nairihar](https://nairihar.medium.com/?source=post_page-----39e83a0a9ded--------------------------------)

·

Follow

Published in

[Bits and Pieces](https://blog.bitsrc.io/?source=post_page-----39e83a0a9ded--------------------------------)

·

3 min read

·

Sep 5, 2023

106



Generated using Leonardo AI

It has already been five years since Node.js introduced the worker\_threads native library, designed for creating workers, essentially functioning as threads.

When it was released, I was incredibly enthusiastic and eager to build something interesting in the scope of that.

When discussing worker threads, I noticed that it’s not straightforward for everyone to build a proper system that would use worker\_threads. Consequently, I embarked on a quest to find abstractions that could simplify and enhance the potential for more intriguing solutions.

I came across several intriguing abstractions, but ultimately, I chose to develop my own — one that is lightweight and straightforward.

**funthreads**

[NPM](https://www.npmjs.com/package/funthreads) | [Github](https://github.com/nairihar/funthreads" \l "readme" \t "_blank)

The idea is simple: you can execute your function in a dedicated thread by utilizing Promises.

import executeInThread from 'funthreads';  
  
async function calculate() {  
 const values = await Promise.all([  
 executeInThread(() => 2 \*\* 10),  
   
 executeInThread(() => 3 \*\* 10)  
 ]);  
   
 console.log(values);  
}  
  
calculate();

You can relocate CPU-intensive operations to separate threads and easily retrieve the results using Promises.

Just try yourself and see.

$ npm i funthreads

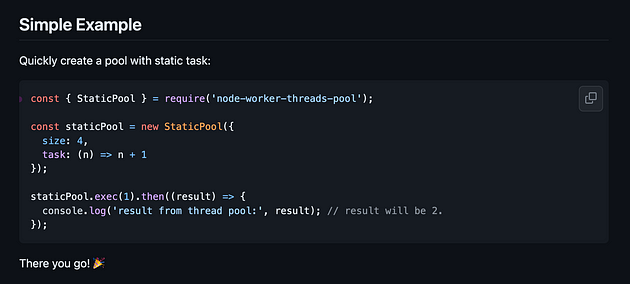
<https://github.com/nairihar/funthreads>

I didn’t spend much time on this library, and though I haven’t used it in a real project, I stayed excited and made small improvements over time.

After publishing this library, I noticed other extended implementations of the same idea.

**node-worker-threads-pool**

As the authors describe: *Simple worker threads pool using Node’s worker\_threads module. Compatible with ES6+ Promise, Async/Await and TypeScript🚀.*



<https://github.com/SUCHMOKUO/node-worker-threads-pool/tree/master>

But this is not all, there is another library that had been released even before worker\_threads.



<https://www.npmjs.com/package/threads>

Previously, they used various strategies and approaches to support multithreading. However, after worker\_threads, their implementations significantly improved.

Here’s a concise example that illustrates how the library operates.

// main.js  
import { spawn, Thread, Worker } from "threads"  
  
const auth = await spawn(new Worker("./workers/auth"))  
const hashed = await auth.hashPassword("Super secret password", "1234")  
  
console.log("Hashed password:", hashed)  
  
await Thread.terminate(auth)

// worker.js  
import sha256 from "js-sha256"  
import { expose } from "threads/worker"  
  
expose({  
 hashPassword(password, salt) {  
 return sha256(password + salt)  
 }  
})

It seems that this is the biggest library that provides a bunch of tools to work with multithreading in Node.js.

P.S. Both of the libraries offer a method to establish a thread pool and employ it for your specific use cases.

Unfortunately, I haven’t used any of them in production, as there wasn’t a need. However, I’m confident that many projects can benefit from them in real-world scenarios.

It’s interesting to observe how they work and understand the mechanics of the abstraction.

funthreads library is quite straightforward, consisting of just a few files with small functions. You can begin with it, and I’m confident that you’ll grasp the concept easily, realizing how simple the provided abstraction is.

**Wrapping up**

Compared with other programming languages in JavaScript everything is quite different. Everything operates in its own unique way and has its proper explanation.

JavaScript has classes, and Node.js supports multithreading. While it may seem overwhelming compared to other languages, it’s important to embrace JavaScript on its own uniqueness.

Thank you for taking the time to read this comprehensive article.  
Feel free to ask any questions or tweet me [@nairihar](https://twitter.com/nairihar)

Also follow my **JavaScript** newsletter on Telegram: [@javascript](https://t.me/javascript)

I hope you found it informative and gained valuable insights from it.