JavaScript is single-threaded. How did it evolve towards a multi-threaded environment and why?

Egor Knyazev

Fontys University of Applied Science

Author Note

[Include any grant/funding information and a complete correspondence address.]

Table of Contents

[Context 3](#_Toc87209258)

[How did JavaScript evolve towards a multi-threaded environment and why? 4](#_Toc87209259)

[Multi/Single threads 4](#_Toc87209260)

[Multi threads benefit 4](#_Toc87209261)

[Origin of JavaScript 6](#_Toc87209262)

[JavaScript usage 6](#_Toc87209263)

[WebAPI 6](#_Toc87209264)

[Dedicated workers 7](#_Toc87209265)

[References 9](#_Toc87209266)

[Footnotes 10](#_Toc87209267)

[Tables 11](#_Toc87209268)

[Codes 12](#_Toc87209269)

[Figures 13](#_Toc87209270)

Context

This research is the part of individual project in which will be final product a multiplayer snake game. App has fronted that based on JavaScript and it behave as multi-threaded but should not. This was the point of starting this research – not clear how it emulates the multi-threading if, in fact, it is approved single-threaded language. In this research will be disassemble work principals of multi-threads behavior JavaScript. The goal of this research paper is to provide information of hidden work and process of above-mentioned functionality. The research areas will be identified first and then teams of two will investigate academic sources that would further our understanding of these topics. This research is carried out with the guideline of DOT framework.

How did JavaScript evolve towards a multi-threaded environment and why?

To answer the main question, I should figure out about part of it, therefore, first logical question of multi-threads implementation is “what are benefits and functionalities it causes?” because, in case of unnecessarily multi-threads, arise contradiction of nature progress. Moreover, we should test the JavaScript how it is acting itself and understood origin of JavaScript. Then I want to dive in parts that realize asynchronous execution of code – how it works and what are responsible for this.

# Multi/Single threads

In a single thread or threading, the process contains only one thread. That thread executes all the tasks related to the process. In this example “Code 1”, the single thread will be result: *A B C*. In a multi-threaded application, multiple threads are executed concurrently. Each thread handles different tasks simultaneously by making optimal use of the resources. Therefore, in this variant result will be *A C B*. Single thread refers to executing an entire process from beginning to end without interruption by a thread while multi-thread refers to allowing multiple threads within a process so that they are executed independently while sharing their resources. Thus, this is the main difference between single thread and multi thread.

## Multi threads benefit

The gain of multithread is we can execute tasks in parallel that is required less time. This is demonstrating “Figure 1” as we can see 2 threads give as twice speed up if not count the constant time that require to assign difference thread. This mean that we can gain speed up nearly amount of threads times, in case, we got sufficient tasks to split them on this amount. Using threads is beneficial for many reasons, but one that is particularly applicable to browsers is that, by offloading CPU-intensive work to a separate thread, the main thread is then able to dedicate more resources to rendering the UI. This can help contribute to a smoother, more user-friendly experience than what might have been traditionally achievable.

# Origin of JavaScript

When Eich created JavaScript in 1995, he created it for Netscape Navigator and it quickly became known as LiveScript. In another quirk of JavaScript history, the team changed the name to JavaScript to reflect Netscape’s support of Java within its browser. The way Eich saw it, higher-end programmers used Java and it might be considered a “component language.” Eich created JavaScript to fill the need for a “glue language” used by informal programmers and designers. This allowed programmers to use JavaScript to put together components and automate

interactions. At this point in our JavaScript history, there were two dominating web browsers: Netscape Navigator (with JavaScript) and Internet Explorer (with Jscript). And by the time the browser world shifted and Internet Explorer became the dominant browser, JavaScript evolved into the endorsed standard for writing interactive processing run in a web browser. For further history details check “Table 1”.

## JavaScript usage

For now, we see that main purpose of JavaScript is orange the communication between the parts of an application. By the testing difference version of JavaScript. I can indicate that before 2005-year version multi-threads not exist. Therefore, there is point that multi-threading is part that works outside the JavaScript. The novation of next versions was a “WebAPI”.

### WebAPI

This part of run-time environment. In another words, this is, literally, not a part of JavaScript but it is a part helps working the JavaScript in the browser. It is a small background scripts that organize work, they responsible for call and callback stacks – at the moment when we put or receive information from the JavaScript in fact we got work with DOM, AJAX and TimeOut. So, when the “Code 1” was executed and come to “SetTimeout()” it has executed not by the JavaScript, it sends to the WebAPI to get executed. Ultimately, when we start executing .js file at browser simultaneously browser start some other parts that assist processing of execution: separate threads to handle garbage collection and other features that don’t need to happen in line with JavaScript execution. In addition, the platform runtimes themselves may use additional threads to provide other features. For instance, the modern Node.js can be configurate to use up to 1024 threads that is controlled by environment variable “UV\_THREADPOOL\_SIZE”.

### Dedicated workers

Web workers allow you to spawn a new environment for executing JavaScript in. JavaScript that is executed in this way is allowed to run in a separate thread from the JavaScript that spawned it. Communication occurs between these two environments by using a pattern called message passing. Recall that it’s JavaScript’s nature to be single threaded. Web workers play nicely with this nature and expose message passing by way of triggering functions to be run by the event loop. It’s possible for a JavaScript environment to spawn more than one web worker, and a given web worker is free to spawn even more web workers. That said, if you find yourself spawning massive hierarchies of web workers, you might need to reevaluate your application. There is more than one type of web worker, the simplest of which is the dedicated worker.

### Completely multi-threaded?

The answer is no. If we execute the “Code 2” we will see that each file had been read one by one. Basically, it is the result that we got only one “Instructor pointer”, “garbage collector”, etc. JavaScript interpreter can execute only one instruction in any time. But, in some cases, we can address to different global object that provides functionality of Threads. This emphasizes one more time that not JavaScipt itself access threads – mostly, multi-threads got the parts that can be highly loaded and splitting work only variant to make and smoothly for user.

## Conclusion

As I research JavaScript from burn until now is a single thread language but we should understand that in modern reality JavaScript not using alone – there were created lots of tools that extends functionality of JavaScript. Due to time, web apps increase in sizes and purposes they have been being used, therefore, the load on programming languages, also, increased. The point of JavaScript responsible how users will interact with app was necessary maintain it fast in an execution. That caused in time JavaScript start covering by extra, sub apps that should assist to make tasks of JavaScript execute conveniently for end-users. Each year tasks had grown in size, thus, new external extends come and come that one moment they become not external part of JavaScript but mutate in the environment of executing it.

References

Lithmee, B. (2019, March 5). *What is the Difference Between Single Thread and Multi Thread in Java*. Pediaa.Com. Retrieved November 4, 2021, from https://pediaa.com/what-is-the-difference-between-single-thread-and-multi-thread-in-java/

@Nikhgoel. (2021, January 21). *Why JavaScript is a single-thread language that can be non-blocking*  GeeksforGeeks. Retrieved November 3, 2021, from <https://www.geeksforgeeks.org/why-javascript-is-a-single-thread-language-that-can-be-non-blocking/>

Team, C. (2021, July 14). *The History of JavaScript [INFOGRAPHIC]*. Checkmarx.Com. Retrieved November 2, 2021, from <https://checkmarx.com/blog/javascript-history-infographic/>

I. I. Thomas Hunter, Bryan English. (2021). *Multithreaded JavaScript* ( First Edition). O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Tables

Table 1

The history of JavaScript

|  |  |  |
| --- | --- | --- |
| Fact | Year | Comment |
| Born | 1995 | Created by Brendan Eich |
| Handed to ECMA | 1996 | Receive official name  ” ECMAScript” |
| ECMAScript 2 is released | 1998 |  |
| ECMAScript 3 is released | 1999 | This release evolved into what is modern JavaScript |
| Community flourishes | 2005 | Releases of jQuery, Prototype, Dojo. Created a term “Ajax” |
| Involved parties come together in Oslo | 2008 |  |
| The CommonJS project is created | 2009 | This resulted in the agreement to drive the language towards using as known as Harmony aka ECMAScript 6 |
| 92% of all websites use JavaScript | 2016 |  |

Note: Today, JavaScript is still everywhere - it’s the most commonly used client-side scripting language. JavaScript is written into HTML documents and enables interactions with web pages in many unique ways. For example, thanks to JavaScript we can automatically schedule appointments and play online games. Furthermore, new developments, such as Node.js, allow the use of JavaScript on server-side while APIs, such as HTML5, allow the control of user media and other device features.

Codes

Code 1

Checking the sequence of output

console.**log**('A');

**setTimeout**(() => {console.**log**('B');}, 3000);

console.**log**('C');

*Notes:* The compiling result will be “A C B”. This example demonstrates an asynchronous behavior of JavaScript

Code 2

Checking executing task order

**import** fs from 'fs/promises';

async **function** getNum(filename) {

**return** **parseInt**(await fs.readFile(filename, 'utf8'), 10);

}

**try** {

**const** numberPromises = [1, 2, 3].**map**(i => getNum(`${i}.txt`));

**const** numbers = await Promise.all(numberPromises);

console.**log**(numbers[0] + numbers[1] + numbers[2]);

} **catch** (err) {

console.error('Something went wrong:');

console.error(err);

}

Figures

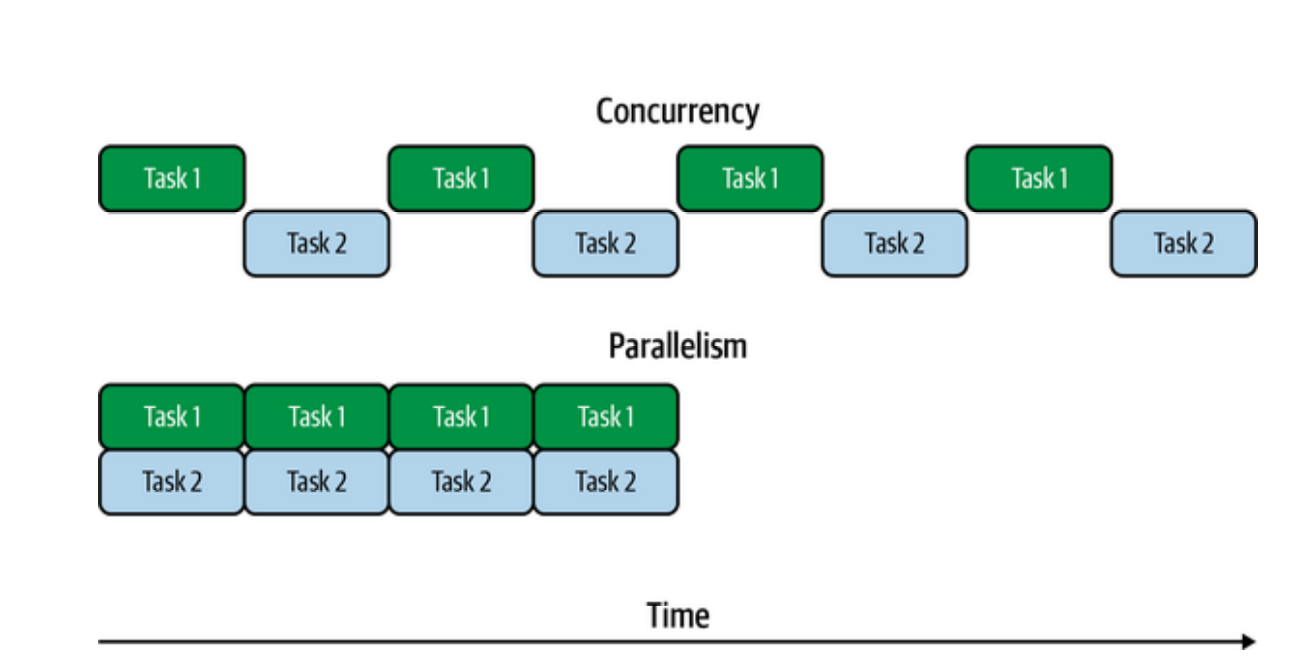


Figure 1. Concurrency versus parallelism; taken from book “*Multithreaded JavaScript*”.