

Working of Asynchronous Functions :

Example of reading a file using 'fs' library of Node.js:

Code:-

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

```
const filePath = "path/to/your/file.txt"; // Replace with the actual path to your file
```

```
fs.readFile(filePath, "utf8", (err, data) => {  
  if (err) {  
    console.error("Error reading the file:", err);  
    return;  
  }  
  
  console.log("File content:", data);  
});
```

Let's take this example. Suppose there's a file named 'chocolate.txt' within the same directory as this JS code file and this file contains a text which is "I want a chocolate!". Now replace this as the file path in `fs.readFile()` function like this `"/chocolate.txt"` and then simulate reading this file and printing the data as said in `console.log("File Content:", data)`. Now, while doing all this I want you to give me stack traces and when and which WebAPIs were called and how the callback went into the callback queue and most importantly at which step were the contents of the file (`chocolate.txt`) were passed onto the callback so that it is able to actually print the written data in that file!

Working:-

Sure, let's go through the step-by-step process, including the stack traces, WebAPI calls, callback queue, and when the contents of the file are passed to the callback. We'll assume the file 'chocolate.txt' with the content "I want a chocolate!" exists in the same directory as the JS code file.

Here's the breakdown:

1. **Initial Setup**:

- The 'fs' module is required using `require("fs")`.
- The 'filePath' variable is defined with the value `"/chocolate.txt"`, representing the path to the file you want to read.

2. **fs.readFile() Call**:

- The `fs.readFile()` function is called with the following arguments:
 - 'filePath': The path to the file (`"/chocolate.txt"`).
 - "utf8": The encoding of the file (UTF-8).
 - A callback function `(err, data) => {...}`.

3. **Callback Registration**:

- The callback function `(err, data) => {...}` is registered to be executed when the file reading operation is complete.
- At this point, the callback is not yet executed; it's just registered for later execution.

4. **WebAPI Call (File Reading)**:

- The `fs.readFile()` function is asynchronous and utilizes the Node.js runtime's underlying file system APIs, which run outside the JavaScript main thread.
- The file reading operation is initiated by the Node.js runtime as a separate task outside the main thread.

5. **Call Stack Emptying**:

- Since the file reading operation is asynchronous, the call stack is empty after the `fs.readFile()` call, all

owing other tasks to continue.

6. ****Callback Execution (File Read Complete)****:

- The file reading operation is completed, and the content of the file ("I want a chocolate!") is available.
- The callback function `(err, data) => {...}` is added to the callback queue.

7. ****Callback Queue and Event Loop****:

- The event loop monitors the callback queue for tasks.
- Since the main stack is empty, the event loop picks up the callback from the queue and pushes it onto the call stack for execution.

8. ****Callback Execution****:

- The callback function is executed, receiving two parameters: `'err'` (which is `'null'`, indicating no error) and `'data'` (which contains the content of the file, "I want a chocolate!").
- The `'console.log("File content:", data);'` statement is executed, printing "File content: I want a chocolate!" to the console.

9. ****Stack Emptying****:

- The callback execution completes, and the call stack is empty again.

Throughout this process:

- The WebAPI used is the Node.js file system API, which handles the file reading operation asynchronously.
- The content of the file ("I want a chocolate!") is passed to the callback function `(err, data) => {...}` when the file reading is complete.
- The `'console.log'` statement inside the callback logs the content to the console.

The asynchronous nature of `'fs.readFile()'` allows other tasks to continue running while the file is being read. Once the file reading is finished, the callback function is executed with the file's content, and it logs the data to the console.