MWA Homework 02 - NodeJS 01

Written Exercises

1. Explain why do we want sometimes to use setImmediate instead of using setTimeout?

setImmediate() is better to use when you want to schedule a function to be executed as soon as possible, but after any currently executing code. This is useful when you want to ensure that a function is executed before any other I/O operations or timers that may be scheduled.

On the other hand, setTimeout() is better to use when you want to schedule a function to be executed after a specific amount of time has passed. This is useful when you want to ensure that a function is executed after a certain delay, regardless of what else is happening in the JavaScript engine.

1. Explain the difference between process.nextTick and setImmediate?

setImmediate() is a method of the global object in Node.js that schedules a callback function to be invoked as soon as the call stack is empty. It is similar to setTimeout(fn, 0), but it is generally considered to be more efficient, as it avoids the overhead of creating a new timer.

process.nextTick() is a method of the process object in Node.js that schedules a callback function to be invoked on the next tick of the event loop. It is similar to setImmediate(), but it has a slightly higher priority, as it is executed before any I/O events or timers.

Both setImmediate() and process.nextTick() are used to schedule callbacks to be invoked as soon as possible, but process.nextTick() is generally used for callbacks that need to be invoked before any other I/O events or timers, while setImmediate() is used for callbacks that can be invoked after any current I/O events or timers have been processed.

1. Name 10 core modules that Node provides by default, and 10 of the Global objects?

**Core modules in Node.js:**

1. http - provides an HTTP server and client
2. https - provides an HTTPS server and client
3. fs - provides a way to interact with the file system
4. path - provides utilities for working with file and directory paths
5. os - provides information about the operating system
6. events - provides an event emitter class for managing events
7. stream - provides a way to work with streaming data
8. util - provides utility functions for various tasks
9. url - provides utilities for working with URLs
10. assert - provides a way to test for and handle errors

**Global objects in Node.js:**

1. process - provides information about the current process
2. global - the global namespace object
3. console - provides a way to output to the console
4. Buffer - provides a way to work with binary data
5. setTimeout - schedules a function to be called after a specified delay
6. setInterval - schedules a function to be called repeatedly with a specified delay
7. clearTimeout - cancels a previously scheduled timeout
8. clearInterval - cancels a previously scheduled interval
9. require - function to include other modules
10. module - provides information about the current module

Exercise 02

Write a function in Node factorial(n) to find asynchronously the factorial of a number. Write your observation (CPU, RAM.. etc) on what happens in Node Process when we calculate the factorial of a large number.

async function factorial(n) {

let product = 1;

for (let i = n; i > 1; i--) {

product \*= i;

await new Promise(resolve => setImmediate(resolve));

}

return product;

}

Observation

This function uses a loop to calculate the factorial of the given number n, and inside the loop it uses await new Promise(resolve => setImmediate(resolve)) to yield control to the event loop on each iteration, allowing other tasks to be executed before continuing with the next iteration.

When calculating the factorial of a large number, the function will take a longer time to complete, and it will also consume more memory to store the intermediate result. This is because the larger the number, the larger the factorial will be and the more memory is needed to store it. Furthermore, since this function is implemented asynchronously, it may not block the execution of other code, and it could run in parallel with other tasks.

Additionally, as the number grows large, the function will also consume more CPU cycles to perform the calculation, this will cause a high consumption in the CPU and RAM, hence it could cause the system to slow down or even crash if the number is too large.