NODEJS INTERNALS

Some code and their output

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

console.log("Hello");

setTimeout(() => console.log("I am set timeout"), 0);

setImmediate(() => console.log("I am set immediate"));

console.log("bye");
```

Output :

```
Hello
bye
I am set timeout
I am set immediate
```

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

console.log("Hello");

setTimeout(() => console.log("I am set timeout"), 0);
setImmediate(() => console.log("I am set immediate"));

// console.log("bye");
```

Output:

```
Hello
I am set immediate
I am set timeout
```

Why does the output change just by removing a console.log?

--> Will answer after understanding the concepts of Nodejs

Concepts of Nodejs -> Working

--> Nodejs runtime environment based on v8 engine
--> Nodejs also uses LibUV to do async I/O tasks
--> LibUV provides -> Thread Pool to Nodejs

Single Thread vs Multi Thread

--> Consider thread to be workers
--> Single thread -> a worker -> can do one task at a time but if it takes time -> can take up other task while first one completes
--> Mutli thread -> a number of workers -> each will do one task -> each will complete task (no matter how much time it takes) -> move on to next task

--> Nodejs -> Single thread -> but uses thread pool to do async tasks

---

--> When we do "node index.js" , node actually starts a process and inside that process is Main thread and a thread pool
--> Main thread -> main worker -> which will do the tasks -> when we say nodejs is single threaded -> we talking about this
--> Inside Main thread:

1. Initialize project
2. Top level Code executed -> require executed first
3. Event callback registered
4. Event Loop starts -> basically a while(true) loop
5. Inside Event Loop are phases -> each phase unique and has a FIFO queue for callbacks
   a. Timer Phase -> system level checks related to timers done -> if something in callback queue -> executed
   b. I/O Pollling Phase -> checks if any IO task completed -> if yes push their callbacks to execute

    c. Run SetImmediate Callbacks
    d. Close Callbacks are run
    e. If nothing pending -> break event loop -> else start from a. again
6. Callbacks in micro task queue -> executed fter each phase -> promise callback -> if present executed after each phase
7. CPU intensive tasks like cryptography, hashing etc --> thread pool workers used

--> Look at this code and try to tell the output based on what we have learnt above

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

console.log("Hello");

setTimeout(() => console.log("I am set timeout"), 0);
setImmediate(() => console.log("I am set immediate"));

fs.readFile("sample.txt", "utf-8", function (err, data) {
        setTimeout(() => console.log("SetTimeout inside FS"), 0);
        setImmediate(() => console.log("Immediate inside FS"));
});

console.log("bye");
```

Output:

```
Hello
bye
I am set timeout
I am set immediate
Immediate inside FS
SetTimeout inside FS
```

--> Try to understand the above given output.

--> Also this is the right time to understand the code and output that we did at the start of this topic.

Understanding Thread Pool size using code

```
const crypto = require("crypto");
const start = Date.now();

crypto.pbkdf2("passsword", "salt1", 100000, 1024, "sha512", (err, data) => {
        console.log(`[${Date.now() - start}ms]: Password 1 hashed`);
});

crypto.pbkdf2("passsword", "salt1", 100000, 1024, "sha512", (err, data) => {
        console.log(`[${Date.now() - start}ms]: Password 2 hashed`);
});

crypto.pbkdf2("passsword", "salt1", 100000, 1024, "sha512", (err, data) => {
        console.log(`[${Date.now() - start}ms]: Password 3 hashed`);
});

crypto.pbkdf2("passsword", "salt1", 100000, 1024, "sha512", (err, data) => {
        console.log(`[${Date.now() - start}ms]: Password 4 hashed`);
});

crypto.pbkdf2("passsword", "salt1", 100000, 1024, "sha512", (err, data) => {
        console.log(`[${Date.now() - start}ms]: Password 5 hashed`);
});

crypto.pbkdf2("passsword", "salt1", 100000, 1024, "sha512", (err, data) => {
        console.log(`[${Date.now() - start}ms]: Password 6 hashed`);
});
```

Output :

```
[1401ms]: Password 1 hashed
[1405ms]: Password 2 hashed
[1422ms]: Password 3 hashed
[1440ms]: Password 4 hashed
[2780ms]: Password 6 hashed
[2783ms]: Password 5 hashed
```

--> this shows -> by default 4 workers in libUV thread pool

--> at the same time 4 are busy -> as long as they are not free -> new cpu intensive task not taken -> so password 5 & 6 not hashed until they free

--> By default libUV workers =4 -> can be changed by changing ->UV_THREADPOOL_SIZE = 6 || WHATEVER_YOU_WANT;
--> This has to be done before starting node -> once process has started -> it can't be done -> do like this -> UV_THREADPOOL_SIZE=6 node index.js

--> Piyush sir did it after the node process was started -> he did process.env.UV_THREADPOOL_SIZE = 6 -> don't know how he did it -> check cohort video at 1:13:28

--> watch piyush sir's video on youtube to understand more : https://www.youtube.com/watch?v=_eJ6KAb56Gw&t=722s

--> read this article --> https://nodejs.org/en/learn/asynchronous-work/event-loop-timers-and-nexttick