TSR

EJERCICIO 2

El siguiente programa NodeJS está inspirado en un ejemplo del Tema 6:

```
 1: const cluster = require('cluster')
 2: const http = require('http')
 3: const interval = 1000
 4: if (cluster.isMaster) {
 5:         let numReqs = 0
 6:         function statistics() {
 7:                 console.log("numReqs=" + numReqs)
 8:         }
 9:         setInterval(statistics, interval)
10:         function messageHandler(msg) {
11:                 if (msg.cmd && msg.cmd == 'notifyRequest')
12:                         numReqs++
13:         }
14:         const numCPUs = require('os').cpus().length
15:         for (let i=0; i < numCPUs/2; i++) {
16:                 let p=cluster.fork()
17:                 p.on('message', messageHandler)
18:         }
19: } else {
20:         http.Server(function(req, res) {
21:                 res.writeHead(200); res.end('hello world\n')
22:                 process.send({ cmd: 'notifyRequest' })
23:         }).listen(8000)
24: }
```

Modifícalo de manera que cada 5 segundos:

- Muestre cuántas peticiones han sido procesadas por todos los trabajadores en esos 5 segundos.(0.2 puntos)
- Si el número de peticiones atendidas en esos 5 segundos es más de 100 veces mayor que el número actual de trabajadores, deberá crearse otro trabajador. (0.2 puntos)

SOLUCIÓN

Una posible solución se da en este programa (son admisibles otras variantes):

```
1:  const cluster = require('cluster')
2:  const http = require('http')
3:  // The new interval should be set to 5 seconds (5000 ms).
4:  const interval = 5000
5:  // We should manage the current amount of workers.
6:  let numWorkers = 0
7:  if (cluster.isMaster) {
8:          let numReqs = 0
9:          function statistics() {
10:                 console.log("numReqs=" + numReqs)
11:                 // Check whether a new worker should be started.
12:                 if (numReqs>numWorkers*100) {
13:                         // If so, create a new worker.
14:                         numWorkers++
15:                         let w=cluster.fork()
16:                         w.on('message', messageHandler)
17:                 }
18:                 // Once we managed the amount of requests, we
19:                 // should reset its value to zero.
20:                 numReqs=0
21:         }
22:         setInterval(statistics, interval)
23:         function messageHandler(msg) {
24:                 if (msg.cmd && msg.cmd == 'notifyRequest')
25:                         numReqs++
26:         }
27:         const numCPUs = require('os').cpus().length
28:         for (let i=0; i < numCPUs/2; i++) {
29:                 // Increase how many workers exist.
30:                 numWorkers++
31:                 let p=cluster.fork()
32:                 p.on('message', messageHandler)
33:         }
34: } else {
35:         http.Server(function(req, res) {
36:                 res.writeHead(200); res.end('hello world\n')
37:                 process.send({ cmd: 'notifyRequest' })
38:         }).listen(8000)
39: }
```

Para responder al primer apartado, hemos modificado el valor de “interval” (línea 4) y hemos añadido la línea 20, que reinicia el valor de numReqs tras haberlo mostrado en la línea 10.

Para implantar el segundo apartado, se ha creado una variable ‘numWorkers’ (línea 6) que mantiene cuántos trabajadores hay. Se incrementa en la línea 30, en el bucle ‘for’ que genera los trabajadores iniciales. Además, se modifica la función ‘statistics’ (líneas 12 a 17). En ese nuevo bloque se comprueba si debe generarse un trabajador adicional. En caso afirmativo, se incrementa ‘numWorkers’, se utiliza cluster.fork() para generarlo y se asocia un *listener* para el evento ‘message’ para gestionar sus mensajes.