SAD: Actividades del Seminario 6

ACTIVIDAD 1: Entender que los "callbacks" no siempre son asincrónicos.

Sólo hay un hilo de ejecución en Node.js. Esto implica que no tendremos por qué preocuparnos sobre la protección de variables compartidas con locks o cualquier otro mecanismo de control de concurrencia.

Sin embargo, hay algunos casos en los que necesitaremos ser cuidadosos.

Un buen principio para razonar sobre la lógica de un programa asincrónico es considerar que TODOS sus callbacks se ejecutarán en un turno posterior al que ahora ejecuta el código que los pasa como argumentos.

Considere el código que se muestra a continuación:

```
var fs = require('fs');
var rolodexFile = fs.open('My rollodex file');
var rolodex = {};

function retrieve(file, name, cb) {
        // Searches for name in file, and
        // invokes cb with record found
}

function processEntry(name, cb) {
        if (rolodex[name]) {
                cb(rolodex[name]);
        }
        else {
                retrieve(rolodexFile, name, function (val) {
                        rolodex[name] = val;
                        cb(val);
                });
        }
}


function test() {
        for (var n in testNames) {
                console.log ('processing ', n);
                processEntry(n, function (res) {
                        console.log('processed ', n);
                })
        }
}
```

```
var testNames = ['a', 'b', 'c'];

test();
```

Cuando sea ejecutado, esperaremos esta salida:

```
processing a
processing b
processing c
processed a
processed b
processed c
```

TODOS los mensajes “processed” aparecen tras TODOS los mensajes “processing”, tal como se esperaba (los “callbacks” parecen ser llamados en turno futuro).

Sin embargo, considere esta variación:

```
var fs = require('fs');
var rolodexFile = fs.open('My rollodex file');
var rolodex = {a: “We know this name”};

function retrieve(file, name, cb) {
        // Searches for name in file, and
        // invokes cb with record found
}

function processEntry(name, cb) {
        if (rolodex[name]) {
                cb(rolodex[name]);
        }
        else {
                retrieve(rolodexFile, name, function (val) {
                        rolodex[name] = val;
                        cb(val);
                });
        }
}


function test() {
        for (var n in testNames) {
                console.log ('processing ', n);
                processEntry(n, function (res) {
                        console.log('processed ', n);
```

```
                    })
            }
}


var testNames = ['a', 'b', 'c'];

test();
```

La salida que obtendríamos sería:

```
processing a
processed a
processing b
processing c
processed b
processed c
```

Observe que ahora NO TODOS los mensajes “processed” se muestran tras TODOS los mensajes “processing”. La razón es que uno de los callbacks ha sido ejecutado EN EL MISMO turno que la función que lo pasó.

Dependiendo de la situación, esto podría introducir problemas difíciles de percibir, en caso de que el código que establece los callbacks y uno o más de los callbacks interfirieran en su acceso a una misma parte del estado del proceso, dejándolo inconsistente.

Esta situación (inconveniente) siempre aparecerá si los callbacks confían en que su invocador preparará sus contextos antes de que ellos inicien su ejecución.

Modifique el programa anterior, utilizando promesas, para garantizar que se respete el orden de ejecución esperado.