Manejo de Archivos (3ra parte): promesas

Teoría

Introducción

Ya vimos que NodeJS ofrece el módulo fs que nos permite operar tanto de forma sincrónica como asincrónica. Así mismo, dentro del paradigma asincrónico, inicialmente ofrecía funciones que reciben un callback para manejar el asincronismo. Con la llegada de las promesas, JS agregó un módulo dentro de fs, que añade versiones de las funciones asincrónicas, pero que en lugar de recibir callbacks, operan mediante promesas. Veremos aquí algunos ejemplos con las mismas funciones sobre las que venimos trabajando. Inicialmente, las promesas se usaron con su sintaxis nativa, y luego se agregó en una versión posterior una sintaxis simplificada utilizando las (entonces) nuevas palabras reservadas “async” y “await”. Nos concentraremos principalmente en está última.

Para no ser excesivamente repetitivo, solamente mostraré un par de ejemplos. El resto de las funciones se puede deducir con facilidad. Caso contrario, dirigirse a la documentación oficial, o consultar con el docente.

Promesas con su sintaxis simplificada (Async/Await):

Leer un archivo

Para leer un archivo usaremos la función `readFile(ruta, encoding)`. Recibe los mismos parámetros que su versión sincrónica.

Al igual que su versión sincrónica, la función se encarga internamente de abrir y cerrar el archivo una vez finalizado su uso.

Ejemplo de uso:

```
async function fun() {
    try {
        const contenido = await fs.readFile('/ruta/al/archivo', 'utf-8')
        console.log(contenido)
    catch (err) {
        // hubo un error, no pude leerlo, hacer algo!
    }
}
```

En el caso de querer hacer algo con la variable fuera del bloque try/catch, la declaración debería hacerse fuera del mismo.

Recordar que debemos anteponer la palabra "await" al llamado a la función para que ésta se comporte de manera bloqueante. Si se omitiera la palabra "await" la instrucción console.log(contenido) se ejecutaría ANTES de que a la variable contenido se le asigne el resultado de la operación de lectura del archivo!

Recordar también que la palabra "await" puede usarse ÚNICAMENTE dentro de una función de tipo "async"

Dado que estas funciones ya no poseen un parámetro que nos permite elegir cómo manejar los errores que pueden surgir de su ejecución, vuelve a ser necesario ejecutarlas utilizando `try / catch` !

Sobreescribir un archivo

Para escribir un archivo usaremos la función `writeFile(ruta, datos)`. Recibe los mismos parámetros que su versión sincrónica, más el callback, que en este caso, solo precisa recibir un parámetro para manejar algún eventual error.

Al igual que su versión sincrónica, la función se encarga internamente de abrir y cerrar el archivo una vez finalizado su uso.

Ejemplo de uso:

```
async function fun() {
    try {
        await fs.writeFile('/ruta/al/archivo', 'TEXTO DE PRUEBA\n')
        console.log('guardado!')
    catch (err) {
        // hubo un error, no pude escribirlo, hacer algo!
    }
}
```

Agregar contenidos a un archivo

Para agregar contenidos a un archivo usaremos la función appendFile`(ruta, datos, callback)`. Recibe los mismos parámetros que su versión sincrónica, más el callback, que en este caso, solo precisa recibir un parámetro para manejar algún eventual error.

Al igual que su versión sincrónica, la función se encarga internamente de abrir y cerrar el archivo una vez finalizado su uso.

Ejemplo de uso:

```
async function fun() {
    try {
        await fs.appendFile('/ruta/al/archivo', 'TEXTO DE PRUEBA\n')
        console.log('agregado!')
    catch (err) {
        // hubo un error, no pude agregarlo, hacer algo!
    }
}
```

Renombrar un archivo

Para escribir un archivo usaremos la función `rename(rutaVieja, rutaNueva, callback)`. Recibe los mismos parámetros que su versión sincrónica, más el callback, que en este caso, solo precisa recibir un parámetro para manejar algún eventual error.

Ejemplo de uso:

```
async function fun(rutaVieja, rutaNueva) {
    try {
        await fs.rename(rutaVieja, rutaNueva)
        console.log('renombrado!')
    catch (err) {
        // hubo un error, no pude renombrarlo, hacer algo!
    }
}
```