TSR: Ejercicios Seminario 1

EJERCICIO 1

ENUNCIADO: Se quiere implementar un servidor **net** que reciba resultados electorales, que le serán transmitidos mediante un número indeterminado de clientes **net**. El servidor deberá contabilizar, acumular adecuadamente, y guardar en ficheros toda la información que le sea transmitida.

Se considera que el servicio a implementar procesará los resultados electorales de unas Elecciones Generales en España. En este caso, la circunscripción electoral es la **provincia**, y en cada provincia hay un número variable, pero elevado, de colegios electorales.

La función de cada cliente **net** será enviar un objeto con los votos obtenidos por cada partido político en un determinado **colegio** electoral. Un par de ejemplos de esta clase de objetos:

{provincia:madrid, colegio:chamberi_14 , pp:3532, psoe:2056, up:3077, cs:1540}

{provincia:barcelona, colegio:sants_12 , pp:105, psoe:143, up:256, cs:196, erc:238}

Los objetos tendrán siempre una propiedad **provincia** que identifica la circunscripción electoral, una propiedad **colegio** que identifica el colegio electoral, y un número variable de propiedades tales que su identificador es el de un partido político y su valor los votos obtenidos por dicho partido.

Estos objetos, serializados con **JSON.stringify**, son los datos que recibirá el servidor **net** a implementar. El servidor ha de procesar los datos recibidos, guardándolos adecuadamente en memoria y en disco:

- En memoria debe mantener una variable de tipo array, llamémosla **votos**, que use la propiedad **provincia** (de los objetos recibidos) como índice del array, y almacene en la posición del array así indexada un objeto con todos los votos recibidos por cada partido en esa provincia.
- En disco, periódicamente (cada 20 segundos), debe guardar un fichero de texto (extensión txt) por cada entrada en el array **votos**. El nombre del fichero será el del índice del array (la propiedad **provincia**) y el contenido del fichero será el valor almacenado en dicha posición del array, serializado con JSON.

A modo de ejemplo, considérese que durante los primeros 20 segundos de ejecución del servidor se han recibido los siguientes datos desde varios clientes **net**:

{provincia:madrid, colegio:chamberi_14 , pp:3500, psoe:2000, up:3000, cs:1500}

{provincia:barcelona, colegio:sants_12 , pp:1000, psoe:1500, up:2000, cs:2000, erc:3000}

{provincia:madrid, colegio:castellana_352 , pp:2000, psoe:3000, up:1000, cs:500}

{provincia:valencia, colegio:vera_sn , pp:2500, psoe:1500, up:2000, cs:2500}

{provincia:madrid, colegio:retiro_8 , pp:4000, psoe:3000, up:2000, cs:2000}

{provincia:barcelona, colegio:provença_115 , psoe:500, up:400, cs:200, erc:300}

Entonces, la variable **votos** del servidor ha de mantener la siguiente información:

votos['madrid'] = {pp:9500, psoe:8000, up:6000, cs:4000}

votos['barcelona'] = {pp:1000, psoe:2000, up:2400, cs:2200, erc:3300}

votos['valencia'] = {pp:2500, psoe:1500, up:2000, cs:2500}

Y en disco se habrán escrito los siguientes ficheros:

Nombre fichero	***Contenido fichero***
madrid.txt	{"pp":9500, "psoe":8000, "up":6000," cs":4000}
barcelona.txt	{"pp":1000, "psoe":2000, "up":2400, "cs":2200, "erc":3300}
valencia.txt	{ "pp":2500, "psoe":1500, "up":2000, "cs":2500}

Se pide implementar el servidor **net**, tomando como base, y manteniendo en la solución, el siguiente código:

```
1  var net = require('net')
2  var fs = require('fs')
3  var votos = [ ]
4
5  var server = net.createServer(function(c) {
6          c.on('data', function(data){
7                  // A COMPLETAR
8          })
9  })
10
11 server.listen(9000,
12   function() { console.log('server bound')
13 })
14
15 function guardar() {
16         // A COMPLETAR
17         console.log('datos volcados a disco')
18 }
19
20 // A COMPLETAR
```

EJERCICIO 2

ENUNCIADO: Se quiere implementar una aplicación que permita obtener resultados electorales a partir de la lectura de un conjunto de ficheros de texto (donde se guardan dichos datos) y desarrollar una sesión interactiva, con el usuario de la aplicación, para consultar resultados de cada circunscripción electoral.

Se considera que la aplicación se ejecuta en un directorio donde se encuentran los ficheros de texto con los resultados electorales. Los nombres y contenidos de los ficheros siguen el formato descrito en el ejercicio anterior, así, por ejemplo:

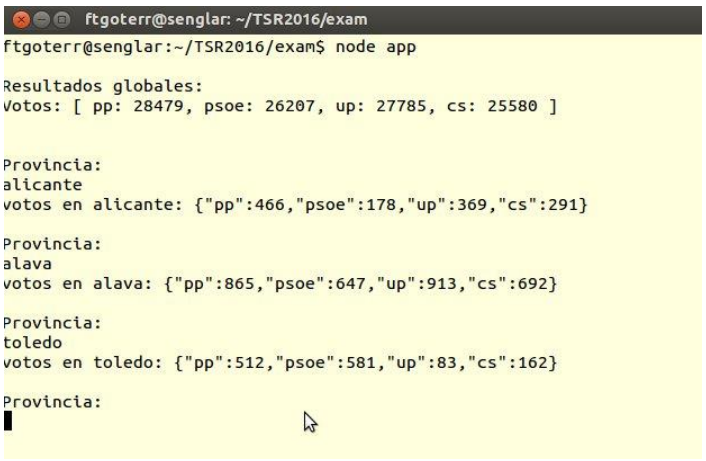
Nombre fichero	***Contenido fichero***
madrid.txt	{“pp”:9500, “psoe”:8000, “up”:6000,” cs”:4000}
barcelona.txt	{“pp”:1000, “psoe”:2000, “up”:2400, “cs”:2200, “erc”:3300}
valencia.txt	{ “pp”:2500, “psoe”:1500, “up”:2000, “cs”:2500}

La aplicación, en primer lugar, debe leer asincrónicamente todos los ficheros, guardando los resultados electorales en un array, llamémoslo **votos**, con el mismo criterio del ejercicio anterior.

La aplicación, en segundo lugar (es decir, una vez leídos todos los ficheros), debe:

- Mostrar los resultados globales (el total de votos obtenidos por cada partido en todas las provincias), guardados en otro array llamado **total_votos**.
- Iniciar una sesión interactiva, mediante **process.stdin**, para consultar los resultados en cada provincia o circunscripción.

La siguiente captura de pantalla sirve como referencia de la funcionalidad de la aplicación:



En este ejemplo, el usuario ha escrito los nombres de 3 provincias (*alicante*, *alava*, *toledo*), y el resto de la salida ha sido generada por la aplicación a implementar.

Se pide implementar la aplicación, tomando como base, y manteniendo en la solución, el siguiente código:

```
1  var fs = require('fs')
2  var total_votos = [ ]
3  var votos = [ ]
4
5  fs.readdir('.', function (err, files) {
6      var count = files.length
7      for (var i=0; i < files.length; i++) {
8          // A COMPLETAR
9      }
10 })
```

Ayuda: Información de API de Node.js

fs.readdir(path[, options], callback)

path <String> | <Buffer>
options <String> | <Object>
encoding <String> default = 'utf8'
callback <Function>

Asynchronous readdir. Reads the contents of a directory. The callback gets two arguments (err, files) where files is an array of the names of the files in the directory excluding '.' and '..'.

---

fs.readFile(file[, options], callback)

file <String> | <Buffer> | <Integer> filename or file descriptor
options <Object> | <String>
encoding <String> | <Null> default = null
flag <String> default = 'r'
callback <Function>

Asynchronously reads the entire contents of a file. The callback is passed two arguments (err, data), where data is the contents of the file. If no encoding is specified, then the raw buffer is returned. If options is a string, then it specifies the encoding.

---

fs.writeFileSync(file, data[, options])

file <String> | <Buffer> | <Integer> filename or file descriptor
data <String> | <Buffer>
options <Object> | <String>
encoding <String> | <Null> default = 'utf8'
mode <Integer> default = 0o666
flag <String> default = 'w'

The synchronous version of fs.writeFile(). Returns undefined.

---

net.createServer([options][, connectionListener])

Creates a new server. The connectionListener argument is automatically set as a listener for the 'connection' event.

---

net.Socket

This object is an abstraction of a TCP or local socket. net.Socket instances implement a duplex Stream interface. They can be created by the user and used as a client (with connect()) or they can be created by Node.js and passed to the user through the 'connection' event of a server.

Event: **'data'**
Emitted when data is received. The argument data will be a Buffer or String. Encoding of data is set by socket.setEncoding(). Note that the data will be lost if there is no listener when a Socket emits a 'data' event.

socket.write(data[, encoding][, callback])
Sends data on the socket. The second parameter specifies the encoding in the case of a string--it defaults to UTF8 encoding.
Returns true if the entire data was flushed successfully to the kernel buffer. Returns false if all or part of the data was queued in user memory.
The optional callback parameter will be executed when the data is finally written out - this may not be immediately.

---

process.stdin

The process.stdin property returns a ***Readable stream*** equivalent to or associated with stdin.

---

stream.Readable

Event: **'data'**

The 'data' event is emitted whenever the stream is relinquishing ownership of a chunk of data to a consumer.

The listener callback will be passed the chunk of data as a string if a default encoding has been specified for the stream using the readable.setEncoding() method; otherwise the data will be passed as a Buffer.