



Módulos node.js

Juan Quemada, DIT - UPM

Módulos node.js

- ◆ node.js incluye un sistema de **módulos** que facilita el diseño de grandes programas
 - Cada modulo es un **fichero diferente** con un **espacio de nombres local**
 - ♦ Un módulo es una closure creada por node, ver <http://nodejs.org/api/modules.html>
- ◆ Un módulo se importa en otro módulo con la función
 - **require(<fichero>)** (ver los detalles en: <http://nodejs.org/api/modules.html>)
- ◆ Un módulo tiene una **parte pública (interfaz)** y otra **privada (implementación)**
 - **Interfaz:** parte visible en el exterior que permite utilizar el módulo a otros
 - ♦ La **interfaz** del módulo se define individualmente los métodos exportados o el objeto interfaz
 - ♦ Los métodos se exportan individualmente con: **exports.metodo_individual = <metodo>**
 - ♦ El objeto interfaz completo se exporta con: **module.exports =<objeto_interfaz>**
 - **Implementación:** código del módulo que crea la funcionalidad
 - ♦ El bloque de instrucciones del módulo (cierre o closure)

Ejemplo de módulos de node.js

◆ El ejemplo siguiente es un programa con dos módulos, cada uno en un fichero diferente

■ Módulo 1: **fichero mi_app.js**

- ◆ importa el otro módulo (en fichero mi_app.js) con: **var circulo = require('./circulo.js');**
- ◆ ambos ficheros están en el mismo directorio, hay que utilizar el path **'./circulo.js'**

■ Módulo 2: **fichero circulo.js**

- ◆ Exporta los dos métodos de la interfaz, area y circunferencia, con **exports.<método> =**

```
// Modulo 1: fichero ejecutable mi_app.js, que importa circulo.js

var circulo = require('./circulo.js'); // al estar en el mismo directorio, el path es ./circulo.js

console.log( 'Area (radio 4): ' + circulo.area(4)); // => Area (radio 4): 50.26548245743669
```

```
// Modulo 2: fichero con librería circulo.js
// -> exporta las propiedades y métodos de la variable global exports

var PI = Math.PI; // variable privada del módulo, no es visible en el exterior del módulo

exports.area = function (r) { return PI * r * r; }; // método de la interfaz
exports.circunferencia = function (r) { return 2 * PI * r; }; // método de la interfaz
```

```

module.exports = function (titulo, inic) {
  var _titulo = titulo;
  var _contenido = inic;

  return {
    titulo: function() { return _titulo; },
    meter: function(nombre, tf) { _contenido[nombre]=tf; },
    tf: function(nombre) { return _contenido[nombre]; },
    borrar: function(nombre) { delete _contenido[nombre]; },
    toJSON: function() { return JSON.stringify(_contenido); }
  }
}

```

```
venus-5:mod01 jq$ node 86-agenda_mod_closure_uso.js
```

```
Teléfono de Jose: 957845123
```

```
Teléfono de Jesús: 978512355
```

```
Teléfono de Pedro: undefined
```

```
Tf de Javier García: 913278561
```

```
Trabajo: {"Javier García":913278561,"José Jimenez":957845123}
```

```
venus-5:mod01 jq$
```

```
var agenda = require('./85-agenda__closure');
```

```

var amigos = agenda ("Amigos",
  { Pepe: 913278561,
    José: 957845123
  });

```

```
amigos.meter("Jesús", 978512355);
```

```

var trabajo = agenda ("Trabajo",
  { "Javier García": 913278561,
    "José Jimenez": 957845123
  });

```

```

console.log('Teléfono de Jose: ' + amigos.tf("José"));
console.log('Teléfono de Jesús: ' + amigos.tf("Jesús"));
console.log('Teléfono de Pedro: ' + amigos.tf("Pedro"));
console.log();
console.log('Tf de Javier García: ' + trabajo.tf("Javier García"));
console.log('Trabajo: ' + trabajo.toJSON());

```

Agenda como modulo
node.js encapsulando
un cierre

```

function Agenda (titulo, inic) {
  this.titulo = titulo;
  this.contenido = inic;
};

Agenda.prototype = {
  titulo: function() { return this.titulo; },
  meter: function(nombre, tf) { this.contenido[nombre]=tf; },
  tf: function(nombre) { return this.contenido[nombre]; },
  borrar: function(nombre) { delete this.contenido[nombre]; },
  toJSON: function() { return JSON.stringify(this.contenido); }
}

```

```
module.exports = Agenda;
```

```
var Agenda = require('./87-agenda_mod_class');
```

```
var amigos = new Agenda ("Amigos",
  { Pepe: 913278561,
    José: 957845123
  });
```

```
amigos.meter("Jesús", 978512355);
```

```
var trabajo = new Agenda ("Trabajo",
  { "Javier García": 913278561,
    "José Jimenez": 957845123
  });
```

```

console.log('Teléfono de Jose: ' + amigos.tf("José"));
console.log('Teléfono de Jesús: ' + amigos.tf("Jesús"));
console.log('Teléfono de Pedro: ' + amigos.tf("Pedro"));
console.log();
console.log('Tf de Javier García: ' + trabajo.tf("Javier García"));
console.log('Trabajo: ' + trabajo.toJSON());

```

```
venus-5:mod01 jq$ node 70-agenda_closure.js
```

```

Agenda:           Amigos
Teléfono de Jesús: 978512355
Teléfono de José:  957845123
Tf de José borrado: undefined

```

```

Agenda:           Trabajo
Tf de Javier García: 913278561
Trabajo:  {"Javier García":913278561,"José Jimenez":957845123}
venus-5:mod01 jq$

```

Agenda como modulo
node.js encapsulando
una clase



RegExp I: Búsqueda de patrones

Expresiones regulares: RegExp

◆ Expresiones regulares:

- ▶ Mecanismo muy eficaz para procesar strings
 - Soportado por muchos lenguajes y herramientas (UNIX)
 - » Emacs, vi, AWK, GREP, PERL, Java, Javascript, Ruby, ...
- ▶ Definen patrones que reconocen cadenas de caracteres específicas
 - Si un patrón reconoce una cadena, se dice que casa (match) con el patrón

◆ En JS se definen en la clase **RegExp** y se pueden crear con

- ▶ Constructor: **RegExp("expresion-regular")**
 - El string puede ser cualquier expresión regular
- ▶ RegExp literal: **/expresion-regular/**
- ▶ Info: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

Búsqueda de patrones

◆ `"string".match(/patrón/)` busca `/patrón/` en `"string"`

▶ Devuelve primera ocurrencia del patrón en un array (`[match]`), y si no casa devuelve `null`

◆ Algunos patrones básicos

- ▶ caracter: `/a/` reconoce solo el caracter "a"
- ▶ secuencia: `/abc/` reconoce la secuencia "abc"
- ▶ principio de string: `/^hoy/` reconoce "hoy" al principio del string
- ▶ final de string: `/hoy$/` reconoce "hoy" al final del string
- ▶ cualquier caracter: `/./` reconoce cualquier caracter

```
"Es hoy".match(/hoy/)      => ['hoy']
```

```
"Es hoy".match(/hoy/)[0]   => 'hoy'
```

```
"Es hoy".match(/hoy$/)     => ['hoy']
```

```
"Es hoy".match(/^hoy/)     => null
```

```
"Es hoy".match(/^..../)    => ['Es h']
```


Clases y rangos de caracteres

- ◆ **Clase de caracteres:** patrón con varios caracteres alternativos entre corchetes
 - ▶ Ejemplo de clase de caracteres: `/[aeiou]/` cualquier vocal (minúscula)
 - ▶ Ejemplo de clase negada: `/[^aeiou]/` no debe ser vocal (minúsc.)
 - ▶ Patrón `\s`: reconoce separadores `[\f\n\r\t\v\u00a0\u1680`]
- ◆ **Rango de caracteres:** Patrón con un rango de caracteres de ASCII alternativos
 - ▶ Rango de caracteres: `/[a-z]/` rango “a-z” de letras ASCII
 - ▶ Patrón `\w`: equivale a `[a-zA-Z0-9_]`
 - ▶ Patrón `\d`: equivale a `[0-9]`

```
"canciones".match(/[aeiou]/)    => ['a']  
"canciones".match(/c[aeiou]/)   => ['ca']  
"canciones".match(/n[aeiou]/)   => ['ne']
```

Controles y **match()**

◆ **match()** admite controles: **i**, **g** y **m**

- **i**: insensible a mayúsculas
- **g**: devuelve array con **todos los “match”**
- **m**: multilínea, **^** y **\$** representan principio y fin de línea

```
"canciones".match(/[aeiou]/g)    => ['a', 'i', 'o', 'e']
```

```
"canciones".match(/c[aeiou]/g)   => ['ca', 'ci']
```

```
"Hoy dice hola".match(/ho/i)     => ['Ho']
```

```
"Hoy dice hola".match(/ho/ig)    => ['Ho', 'ho']
```

```
"Hola Pepe\nHoy vás".match(/^Ho/g) => ['Ho']
```

```
"Hola Pepe\nHoy vás".match(/^ho/gim) => ['Ho', 'Ho']
```



RegExp II: Repetición y alternativa

Operadores de Repetición

◆ + (una o más veces):	/a+/	casa con:	"a", "aa", "aaa", ..
◆ ? (cero o una vez):	/a?/	casa solo con:	" " y "a"
◆ * (cero o más veces):	/a*/	casa con:	"", "a", "aa", "aaa", ...
◆ {n} (n veces):	/a{2}/	casa solo con:	"aa"
◆ {n,} (n o más veces):	/a{2,}/	casa con:	"aa", "aaa", "aaaa", ...
◆ {n,m} (entre n y m veces):	/a{2,3}/	casa solo con:	"aa" y "aaa"

"tiene".match(/[aeiou]+/g) => ['ie', 'e'] // cadenas no vacías de vocales

"tiene".match(/[aeiou]?/g) => ['', 'i', 'e', '', 'e', ''] // vocal o nada

"tiene".match(/[aeiou]*/g) => ['', 'ie', '', 'e', ''] // cadenas de vocales (incluyendo "")

"Había un niño.".match(/[a-zñáéíóú]+/ig) => ['Había', 'un', 'niño']

// casa con palabras en castellano: ascii extendido con ñ, á, é, í, ó, ú

Repetición ansiosa o perezosa

- ◆ Los operadores de repetición son “ansiosos” y reconocen
 - ▶ la **cadena más larga posible que casa con el patrón**
- ◆ Pueden volverse “perezosos” añadiendo “?” detrás
 - ▶ Entonces reconocen la **cadena más corta posible**

<code>"aaabb".match(/a+/)</code>	<code>=> ['aaa']</code>
<code>"aaabb".match(/a+?/)</code>	<code>=> ['a']</code>
<code>"ccaaccbccaa".match(/.+cc/)</code>	<code>=> ['ccaaccbcc']</code>
<code>"ccaaccbccaa".match(/.+?cc/)</code>	<code>=> ['ccaacc']</code>

Patrones alternativos

- ◆ “|” define dos patrones alternativos, por ejemplo
 - ▶ `/[a-z]+/` casa con palabras escritas con caracteres ASCII
 - ▶ `/[0-9]+/` casa con números decimales
 - ▶ `/[a-z]+|[0-9]+/` casa con palabras o números

```
"canciones".match(/ci|ca/)      => [ 'ca' ]
```

```
"canciones".match(/ci|ca/g)     => [ 'ca','ci' ]
```

```
"1 + 2 --> tres".match(/[a-z]+|[0-9]+/g)  => ['1', '2', 'tres']
```



RegExp III: Subpatrones y sustituciones

Subpatrones

- ◆ Dentro de un patrón podemos delimitar subpatrones
 - ▶ Un subpatrón es una parte del patrón delimitada entre paréntesis
- ◆ Por ejemplo `/(c)([aeiou])` tiene dos subpatrones
 - ▶ `(c)` es el primer subpatrón
 - ▶ `([aeiou])` es el segundo subpatrón y así sucesivamente
- ◆ **"string".match(/patrón/)** busca patrón y subpatrones en el string
 - ▶ Devuelve array: `[match, match_subpatron_1, match_subpatron_2, ...]`

```
"canciones".match(/(c)([aeiou])/)    => ['ca','c', 'a']
```

```
"canciones".match(/c([aeiou])n/)    => ['can', 'a']
```

```
"canciones".match(/(..)..(..)/)      => ['cancio', 'ca', 'io']
```


Sustitución de patrones

- ◆ La clase String tiene el **método replace()** para sustituir patrones
- ◆ La expresión: `"string".replace(/patrón/, x)`
 - ▶ devuelve "string" sustituyendo el primer match de "patrón" por x
- ◆ El patrón también puede tener controles i, g y m
 - ▶ i: insensible a mayúsculas
 - ▶ g: devuelve string con todos los “match” sustituidos
 - ▶ m: multilínea, ^ y \$ representan principio y fin de línea

`"Número: 142719".replace(/1/, 'x')` \Rightarrow `'Número: x42719'`

`"Número: 142719".replace(/1/g, 'x')` \Rightarrow `'Número: x427x9'`

`"Número: 142719".replace(/[0-9]+/, '<número>')` \Rightarrow `'Número: <número>'`

Sustitución con subpatrones

- ◆ Dentro de un patrón podemos delimitar subpatrones
 - ▶ Un subpatrón se delimita con paréntesis
- ◆ Por ejemplo `/([ae]+)([iou]*)/` tiene dos subpatrones
 - ▶ `$1` representa el match del primer subpatrón
 - ▶ `$2` el match del segundo y así sucesivamente
 - ▶ `$3`

```
"Número: 142,719".replace(/([0-9]+)([0-9]*)?/, '$1')    => 'Número: 142'
```

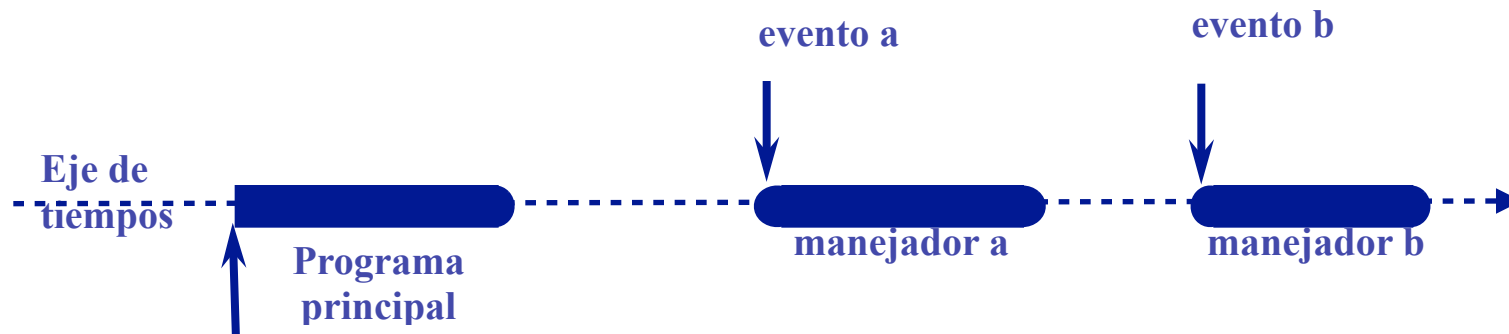
```
"Número: 142,719".replace(/([0-9]+)([0-9]*)?/, '0$2')    => 'Número: 0,719'
```

```
"Número: 142,719".replace(/([0-9]+)([0-9]*)?/, '$1.$2')    => 'Número: 142.719'
```

Eventos node.js

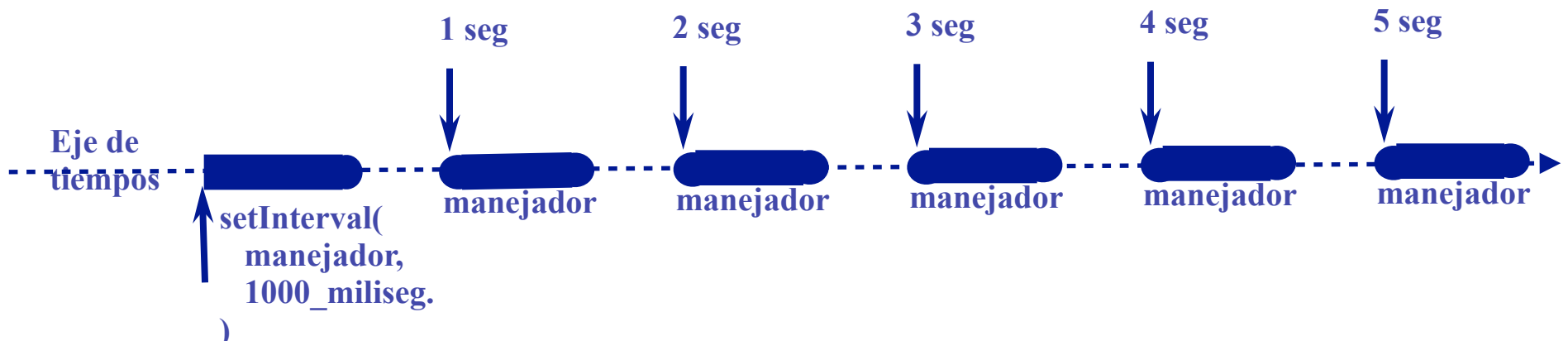
Eventos y Manejadores

- ◆ JavaScript utiliza eventos para interaccionar con el entorno
 - Hay muchos tipos: Temporizadores, métodos HTTP, datos, ...
- ◆ Un evento se programa al instalar un manejador (callback)
 - El manejador es una función que se ejecuta al ocurrir el evento
- ◆ El script inicial programa los eventos iniciales
 - Estos se ejecutan al ocurrir los eventos programados
 - ◆ Estos pueden programar nuevos eventos, si fuesen necesarios

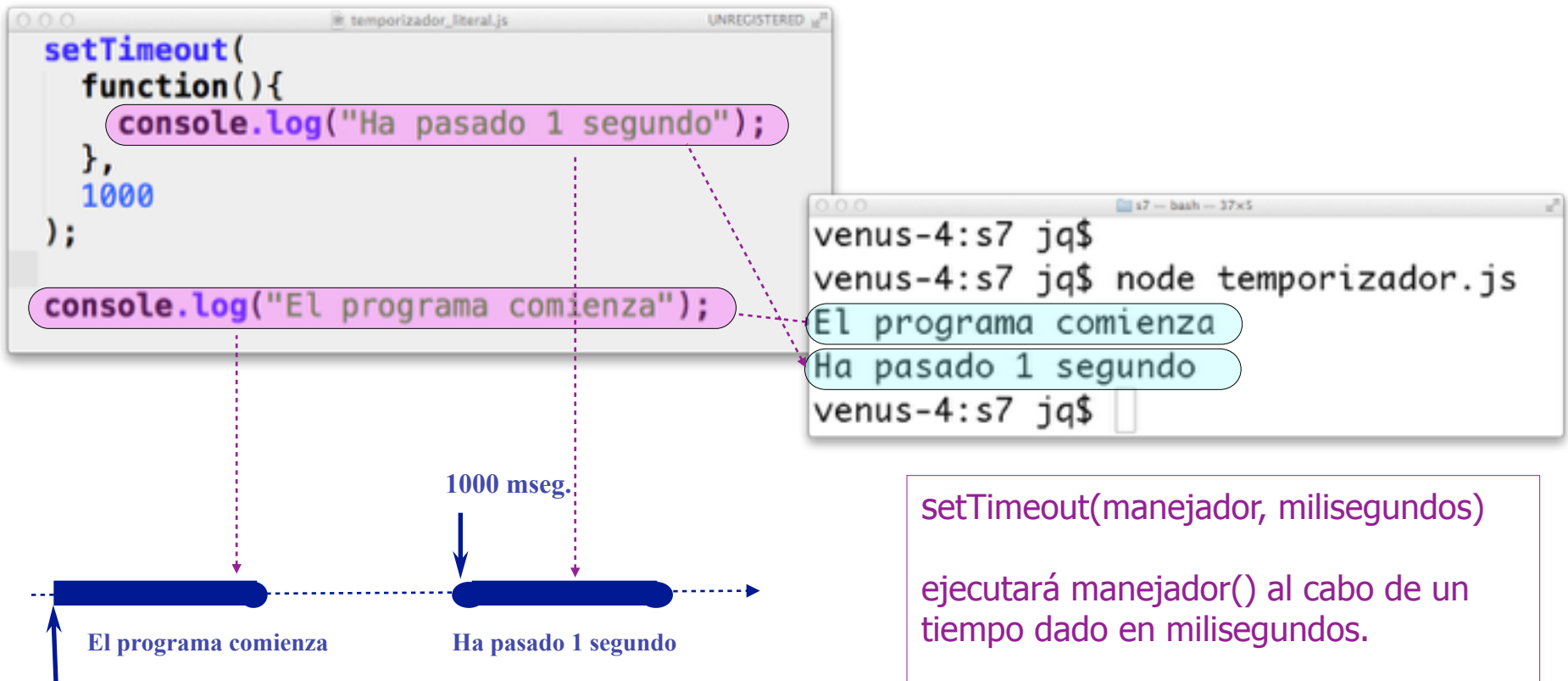


Eventos periódicos con setInterval(...)

- ◆ JavaScript tiene una función **setInterval (..)**
 - para programar eventos periódicos
- ◆ **setInterval (manejador, periodo_en_milisegundos)**
 - tiene 2 parámetros
 - ◆ **manejador**: función que se ejecuta al ocurrir el evento
 - ◆ **periodo_en_milisegundos**: tiempo entre eventos periódicos



Ejemplo con temporizador II

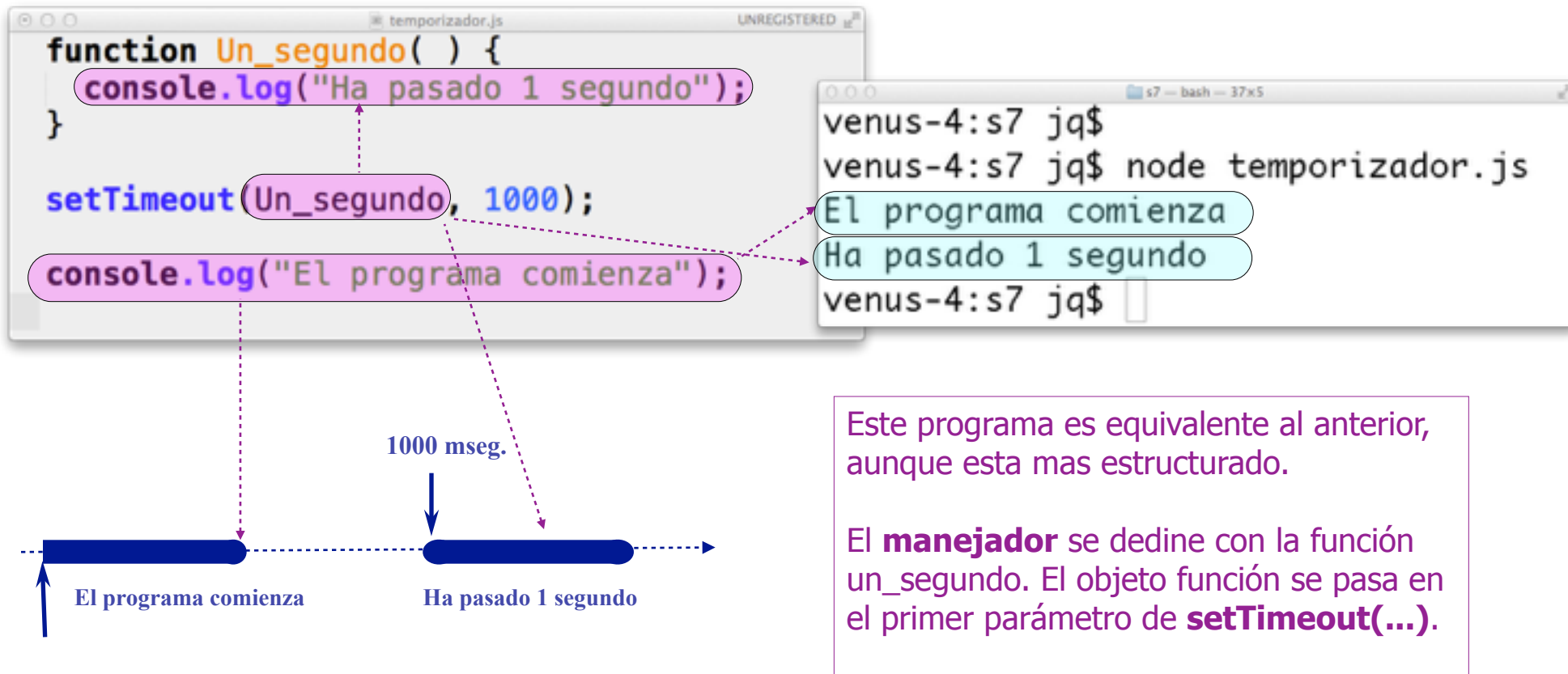


`setTimeout(manejador, milisegundos)`

ejecutará `manejador()` al cabo de un tiempo dado en milisegundos.

El **manejador** se crea con un **literal de función** directamente en el primer parámetro de **`setTimeout(...)`**.

Ejemplo con temporizador



Eventos: Clase EventEmitter

◆ Las clases que emiten eventos derivan de **EventEmitter**

- Heredando métodos como **on(...)** (**addListener(...)**), **emit(...)**, ..
 - Documentación en: <http://nodejs.org/api/events.html>

◆ Un manejador (callback) se define con método **on** (o **addListener**)

- **obj.on('evento', function (params) {.. <código de manejador> ..})**
 - que añade el manejo al array de eventos del objeto
 - El método **on(..)** es totalmente equivalente a **addListener(..)**

◆ El método **obj.emit(<evento>, <p1>, <p2>, ...)**

- envía **<evento>** al objeto **obj**
 - pasando los parámetros <p1>, <p2>, ... al manejador
 - El evento se atenderá por un manejador de dicho objeto, si existe y no afecta a otros objetos.

◆ Un evento tiene por lo tanto 3 elementos asociados

- **nombre**, **manejador** (callback) y **objeto** asociado



Entorno de ejecución de node.js

Objetos global y process

comando node

```
venus-2:~ jq$ node --help
Usage: node [options] [ -e script | script.js ] [arguments]
       node debug script.js [arguments]

Options:
  -v, --version          print node's version
  -e, --eval script      evaluate script
  -p, --print            print result of --eval
  -i, --interactive      always enter the REPL even if stdin
                        does not appear to be a terminal
  --no-deprecation       silence deprecation warnings
  --trace-deprecation    show stack traces on deprecations
  --v8-options           print v8 command line options
  --max-stack-size=val  set max v8 stack size (bytes)

Environment variables:
NODE_PATH               ':'-separated list of directories
                        prefixed to the module search path.
NODE_MODULE_CONTEXTS    Set to 1 to load modules in their own
                        global contexts.
NODE_DISABLE_COLORS     Set to 1 to disable colors in the REPL

Documentation can be found at http://nodejs.org/
venus-2:~ jq$
```

◆ **node:** es un comando UNIX

- Arranca un proceso con un intérprete de JavaScript que ejecuta el programa
- El entorno es ahora el sistema operativo y no el navegador

- HOME
- DOWNLOAD
- ABOUT
- NPM REGISTRY
- DOCS**
- BLOG
- COMMUNITY
- LOGOS
- JOBS



Librería node.js

<http://nodejs.org/api/>

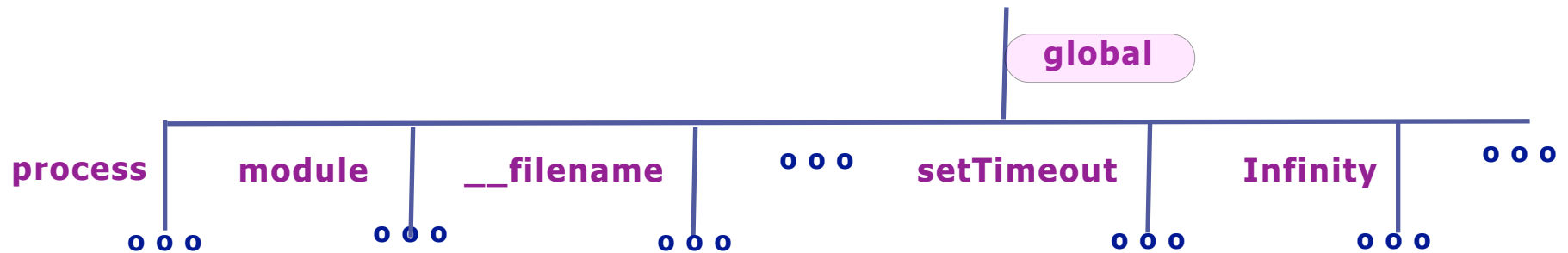
Node.js v0.10.25 Manual & Documentation

[Index](#) | [View on single page](#) | [View as JSON](#)

Table of Contents

- [About these Docs](#)
- [Synopsis](#)
- [Assertion Testing](#)
- [Buffer](#)
- [C/C++ Addons](#)
- [Child Processes](#)
- [Cluster](#)
- [Console](#)
- [Crypto](#)
- [Debugger](#)
- [DNS](#)
- [Domain](#)
- [Events](#)
- [File System](#)
- [Globals](#)
- [HTTP](#)
- [HTTPS](#)
- [Modules](#)
- [Net](#)
- [OS](#)
- [Path](#)
- [Process](#)
- [Punycode](#)
- [Query Strings](#)
- [Readline](#)
- [REPL](#)
- [Stream](#)
- [String Decoder](#)
- [Timers](#)
- [TLS/SSL](#)
- [TTY](#)
- [UDP/Datagram](#)
- [URL](#)
- [Utilities](#)
- [VM](#)
- [ZLIB](#)

Objeto global (this)



◆ El entorno de ejecución de JavaScript en **node** es el **objeto global**

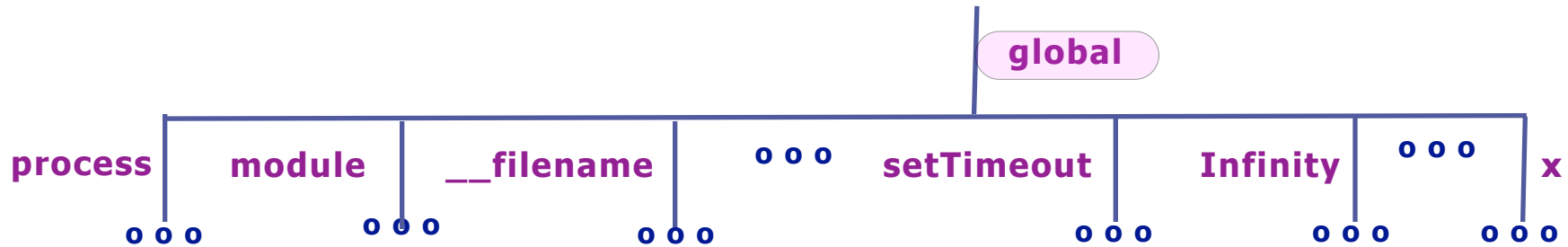
- El **objeto global** tiene propiedades con información sobre
 - ◆ Objetos predefinidos de JavaScript, el **proceso UNIX** en que se ejecuta node,

◆ JavaScript permite referenciar **global** como **this** u omitirse

- La **propiedad process** de **global** se puede referenciar como
 - ◆ **global.process**, **this.process** o **process**

◆ Documentación: <http://nodejs.org/api/globals.html>

Propiedades globales y entorno de ejecución



◆ Un programa JavaScript se ejecuta con el objeto **global** como entorno

- una asignación a una variable no definida como `x = 1;`
 - ◆ Crea una nueva **propiedad de global** de nombre `x`, porque
 - `x = 1;` es equivalente a `this.x = 1;` y a `global.x = 1;`

◆ **Olvidar definir una variable**, es un **error muy habitual**

- y al **asignar un valor a la variable no definida**, JavaScript no da error
 - ◆ sino que crea una **nueva propiedad del entorno global**
- Es un **error de diseño de JavaScript** y hay que tratar de evitarlo

Módulo Process

◆ Node se ejecuta en un proceso del S.O. (UNIX)

- La propiedad **global.process**
 - es la interfaz con el proceso y el S.O. desde un programa node.js
 - Documentación: <http://nodejs.org/api/process.html>

◆ El módulo process incluye

- Parámetros e identificadores del proceso
 - Incluida la cola de eventos de **node.js**
- Métodos de manejo del proceso
 - Por ejemplo: **process.exit()** termina la ejecución del proceso

◆ process es una instancia de EventEmitter

- conecta eventos y variables JavaScript con comandos del S.O.
 - <http://nodejs.org/api/events.html>

argv

◆argv – propiedad de process

- Array de argumentos
 - Orden: node, filename, arg1, arg2, ...

```
e8-argv.js UNREGISTERED
var i;
for(i = 0; i < process.argv.length; i++) {
  console.log('Parametro ' + i + " = " + process.argv[i]);
}
```

```
t3 — bash — 55x6
venus:t3 jq$ node argv.js param1 param2
Parametro 0 = node
Parametro 1 = /Users/jq/Desktop/y-core12-13/t3/argv.js
Parametro 2 = param1
Parametro 3 = param2
venus:t3 jq$
```

Errores y Excepciones

```
e58-caughtException copy.js  UNREGISTERED e2
console.log('Este mensaje saldrá por consola');

funcionIndefinida(); // Función indefinida -> Error

console.log('Este mensaje NO sale por consola');
```

◆ Errores y excepciones interrumpen el programa

```
venus-4:s7 jq$
venus-4:s7 jq$
venus-4:s7 jq$ node e59-error.js
Este mensaje saldrá por consola

/Users/jq/Desktop/MOOC_FirefoxOS/s7/e59-error.js:3
funcionIndefinida(); // Función indefinida -> Error
^
ReferenceError: funcionIndefinida is not defined
    at Object.<anonymous> (/Users/jq/Desktop/MOOC_FirefoxOS/s7/e59-error.js:3:1)
    at Module._compile (module.js:449:26)
    at Object.Module._extensions..js (module.js:467:10)
    at Module.load (module.js:356:32)
    at Function.Module._load (module.js:312:12)
    at Module.runMain (module.js:492:10)
    at process.startup.processNextTick.process._tickCallback (node.js:244:9)
venus-4:s7 jq$
```


uncaught-Exception

```
e57-uncaughtException.js  UNREGISTERED
// Manejador del evento: uncaughtException
process.on('uncaughtException', function(err) {
  console.log('PROGRAMA ABORTADO: ERROR:\n  -> ' + err);
});

console.log('Este mensaje saldrá por consola');

funcionIndefinida(); // Función indefinida -> Error

console.log('Este mensaje NO saldrá por consola');
```

◆ uncaughtException

- Excepción o error no capturado por ningún manejador

```
s7 — bash — 55x6
venus-4:s7 jq$
venus-4:s7 jq$ node e57-uncaughtException.js
Este mensaje saldrá por consola
PROGRAMA ABORTADO: ERROR:
  -> ReferenceError: funcionIndefinida is not defined
venus-4:s7 jq$
```

uncaught-Exception

```
e58-caughtException.js  UNREGISTERED
console.log('Este mensaje saldrá por consola');

try {
  funcionIndefinida(); // Función indefinida -> Error
}
catch (err) {
  console.log('ERROR CAPTURADO: \n -> ' + err);
};

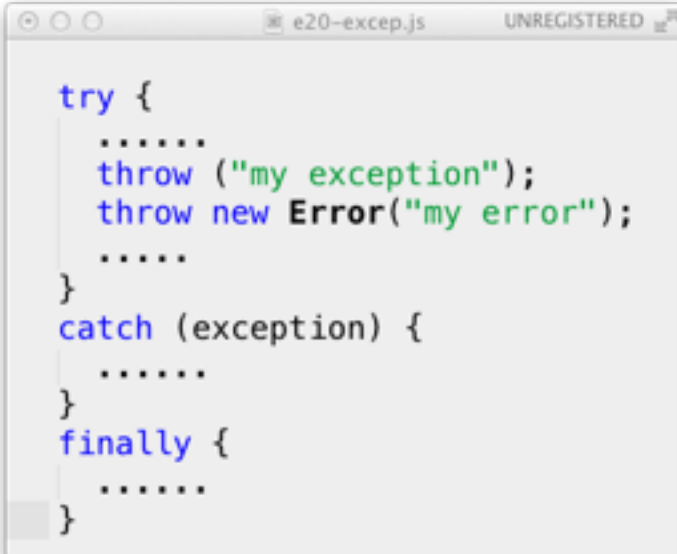
// ERROR capturado, programa sigue
console.log('Este mensaje SI sale ahora por consola');
```

- ◆ Captura del error con la sentencia try/catch
 - Catch captura el error y el programa sigue

```
s7 — bash — 56x7
venus-4:s7 jq$
venus-4:s7 jq$ node e58-caughtException.js
Este mensaje saldrá por consola
ERROR CAPTURADO:
  -> ReferenceError: funcionIndefinida is not defined
Este mensaje SI sale ahora por consola
venus-4:s7 jq$
```

Errores, excepciones y try/catch/finally

- ◆ Las excepciones **interrumpen la ejecución** de un programa
 - Salvo si **se capturan** dentro de try en una sentencia **try/catch/finally**
 - ♦ **catch** captura cualquier excepción o error producido dentro de try
 - la parte **finally** se ejecuta siempre, haya excepción o no
- ◆ La sentencia **throw** lanza
 - excepciones o errores
 - ♦ **Error**: excepción con **objeto Error**
- ◆ Producen **errores**
 - Invocación de métodos en un objeto
 - ♦ que **no tiene definido el método**
 - Utilización de **variables no definidas**
 - **Errores de sintaxis**

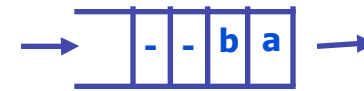


```
try {
    .....
    throw ("my exception");
    throw new Error("my error");
    .....
}
catch (exception) {
    .....
}
finally {
    .....
}
```

Gestión de la concurrencia en node.js

El bucle de eventos

Cola de Eventos

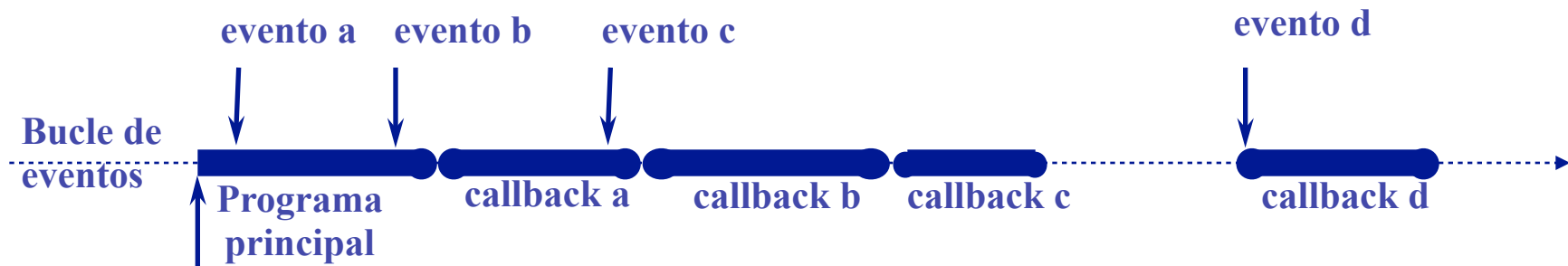


♦ node.js se ejecuta en un **único hilo (thread) de ejecución**

- Al arrancar el proceso, el hilo ejecuta primero el **programa principal**
 - Después **atiende a los eventos** que llegan a la **cola de eventos**
 - Ejecutando sus **manejadores** (o callbacks)

♦ node.js no consume recursos extra mientras no hay eventos que atender

- El resto de actividades del S.O. se puede ejecutar sin problemas
 - node finaliza cuando no hay ningún manejador (callback) de evento programado



Ejemplo: nextTick()

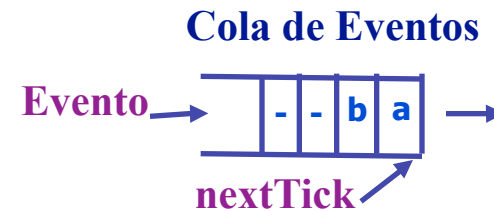
◆ nextTick(<callback>)

- Método de **process**
 - Introduce <callback> adelantando a los otros eventos
- **nextTick**: estrategia FIFO

◆ setTimeout() con retardo “0”

- entra en cola inmediatamente

```
t3 -- bash -- 30x7
venus:t3 jq$ node nextTick.js
1 -> Fin de Programa Principal
2 -> Tick D
3 -> Tick E
4 -> Evento B
5 -> Evento A
venus:t3 jq$
```



```
nextTick.js  UNREGISTERED
5 -> setTimeout(function() { console.log('Evento A'); }, 2);
4 -> setTimeout(function() { console.log('Evento B'); }, 0);

2 -> process.nextTick(function() { console.log('Tick D'); });
3 -> process.nextTick(function() { console.log('Tick E'); });
1 -> console.log('Fin de Programa Principal');
```

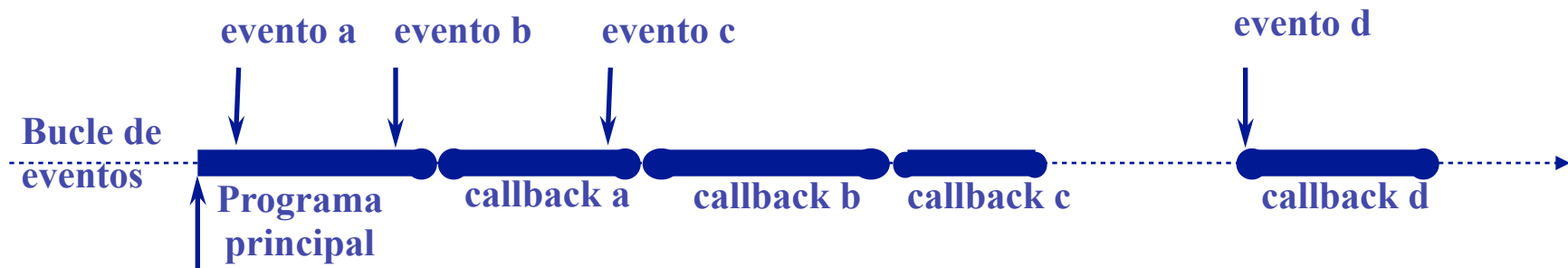
node.js garantiza exclusion mutua

♦ node es muy sencillo de programar

- Los manejadores de eventos se ejecutan en serie

♦ La gestión de la cola de eventos

- garantiza exclusión mutua en el acceso a variables y objetos
 - No se necesitan mecanismos de exclusión mutua: zonas críticas, monitores, ...



Ejemplo de Exclusión Mutua

♦ node.js: la concurrencia se basa en callbacks de atención a eventos

♦ Un callback nunca se interrumpe antes de finalizar su ejecución

- Se ejecutan en serie

♦ Esto **garantiza** que un callback deja las variables comunes en **un estado consistente**

```
mutual_exclusion.js
var tiempo = 6000; // variable compartida

setInterval(function() {
    tiempo = tiempo - 1000;
    console.log('A consume: 1000');
    if (tiempo < 1) {process.exit()};
},
1000
);

setInterval(function() {
    tiempo = tiempo - 1000;
    console.log('B consume: 1000');
    if (tiempo < 1) {process.exit()};
},
1000
);
```

```
t3 - bash - 38x8
venus:t3 jq$ node mutual_exclusion.js
A consume: 1000
B consume: 1000
A consume: 1000
B consume: 1000
A consume: 1000
B consume: 1000
venus:t3 jq$
```

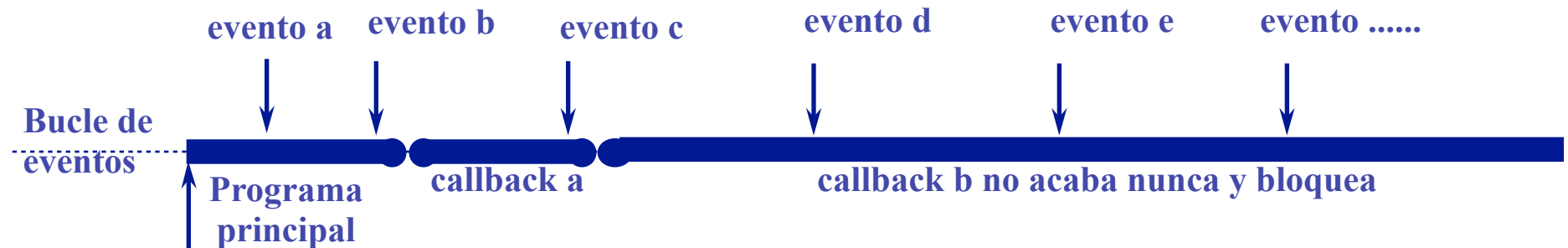

Bloqueo en node.js

◆Bloqueo

- **Problema importante** de la programación concurrente
 - Un programa, o parte de él, **deja de ejecutarse**, esperando que otro acabe

◆Programa principal y manejadores de node.js

- pueden bloquear al resto **solo por inanición** (“starvation”)
 - **Deben finalizar** para que **otros eventos puedan atenderse**
- Si algún Callback **no finalizase, no se atenderán** mas eventos
 - y el servidor **se bloquea**



Servidores node.js

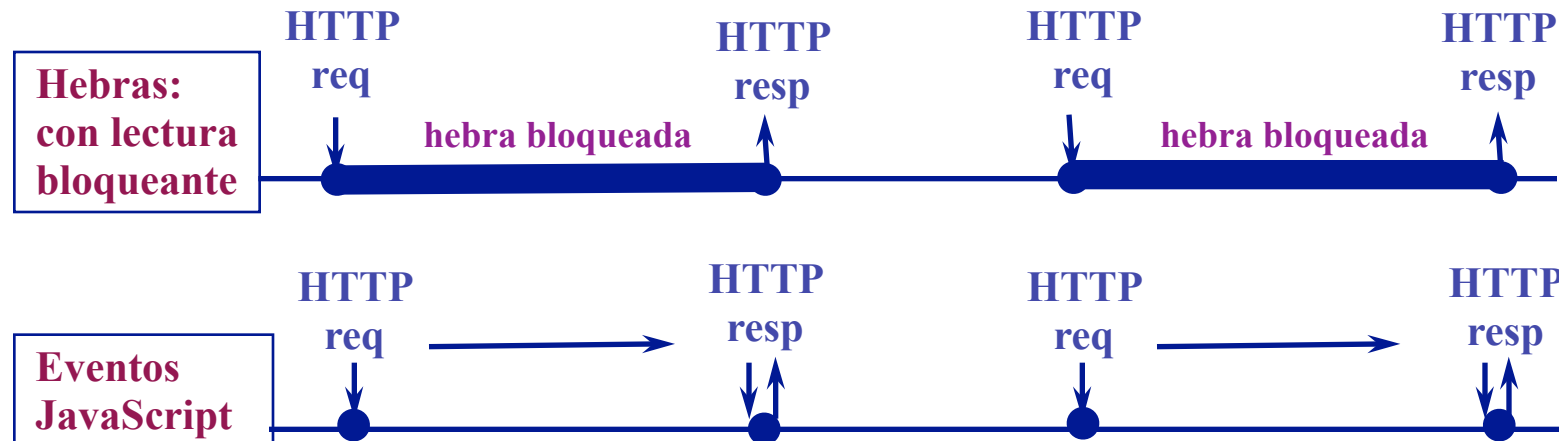
◆ Servidor tradicional (hebras)

- 1 hebra por petición de cliente
 - consume muchos recursos
- **Servidores más lentos**
 - Apache, TOMCAT, PHP, RoR, Django, ..

◆ node.js + express.js (eventos)

- 1 callback atiende muchos clientes
 - **Servidores muy eficientes**
- **node.s**: 10-100 veces más eficiente

The cost of I/O		
Instruc. acceso a memoria	L1-cache	3 cycles
	L2-cache	14 cycles
	RAM	250 cycles
<hr/>		
E/S con acceso en paralelo	Disk	41 000 000 cycles
	Network	240 000 000 cycles



Acceso a Ficheros

Módulo File System

◆ El módulo file system

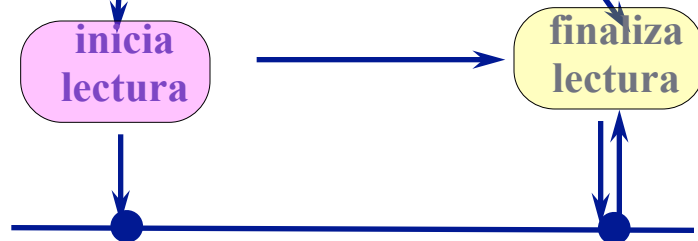
- Da acceso al sistema de ficheros
 - Documentación: <http://nodejs.org/api/fs.html>
- Métodos
 - Ficheros: **open, read, write, append, rename, close, ...**
 - Directorios: **readdir, rmdir, exists, stats,**
 - Gestionar permisos: **chown, chmod, fchown, lchown, ...**
 - Enlaces simbólicos: **link, symlink,**
 -

◆ Hay que importar el módulo antes de utilizarlo

- **require('fs')**

Lectura de un fichero

```
1 var fs = require('fs');
2
3 fs.readFile('file.js',
4             'ascii',
5             function(err, data) {
6                 console.log(data)
7             }
8 );
```



```
venus:t3 jq$ node file.js
var fs = require('fs');

fs.readFile('file.js',
            'ascii',
            function(err, data) {
                console.log(data)
            }
);
venus:t3 jq$
```

◆ Programa principal

- **Carga biblioteca “File System”**
 - “**var fs = require(‘fs’)**”
 - Da acceso al sistema de ficheros
- **Arranca lectura del fichero**
 - **readFile(...);**
 - ‘file.js’: nombre de fich.
 - ‘ascii’: formato
 - **function(..){..}**: callback

◆ Callback

Ejemplo: Copy



- ◆ Importa 'fs'
- ◆ Comprueba params
- ◆ Copia ficheros
 - <orig> a <dest>

```
var fs = require('fs'); // importa módulo file system

if (process.argv.length != 4){ // parámetros mal?
  console.log('syntax: "node copy <orig> <dest>"');
  process.exit() // finaliza proceso node
}

fs.readFile(
  process.argv[2], // fichero <orig>
  function(err, data) { // callback de finalización
    fs.writeFile(
      process.argv[3], // fichero <dest>
      data, // contenido de <orig>
      function (err) { // callback de finalización
        if (err) throw err;
        console.log('file copied');
      }
    );
  }
);
```

lectura de fichero → Escritura de fichero → Final de escritura

```
venus:t3 jq$ node copy.js
syntax: "node copy <orig> <dest>"
venus:t3 jq$ node copy.js bb
syntax: "node copy <orig> <dest>"
venus:t3 jq$ node copy.js bb cc
file copied
venus:t3 jq$
```

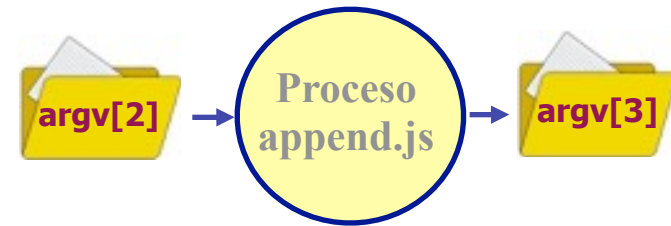
Ordenación de Callbacks en serie

- ♦ La anidación de **callbacks** garantiza orden de ejecución
 - Es un patrón de programación muy habitual en node.js

```
fs.readFile(           // Comienza lectura de fichero
  process.argv[2],
  function(err, data) { // callback de finalización
    fs.writeFile(       // Comienza escritura de fichero
      process.argv[3],
      data,
      function (err) {  // callback de finalización
        if (err) throw err; // Final de escritura
        console.log('file copied');
      }
    );
  }
);
```



Ejemplo: Append



```
var fs = require('fs'); // importa módulo file system

if (process.argv.length !== 4){ // parámetros mal?
  console.log('syntax: "node append <orig> <dest>"');
  process.exit() // finaliza proceso node
}

fs.readFile(
  process.argv[2], // fichero <orig>
  function(err, data) { // callback de finalización
    fs.appendFile(
      process.argv[3], // fichero <dest>
      data, // contenido de <orig>
      function (err) { // callback de finalización
        if (err) throw err;
        console.log('files appended');
      }
    );
  }
);
```

- ◆ Importa 'fs'
- ◆ Comprueba params
- ◆ Concatena ficheros
 - Añade <orig>
 - al final de <dest>

```
venus:t3 jq$ node append.js bb
syntax: "node append <orig> <dest>"
venus:t3 jq$ node append.js bb cc
files appended
venus:t3 jq$
```


Flujos (streams)

Streams de node.js

◆Stream: interfaz para flujos continuos de datos

- Es una instancia de EventEmitter
 - Modulo Stream: <http://nodejs.org/api/stream.html>

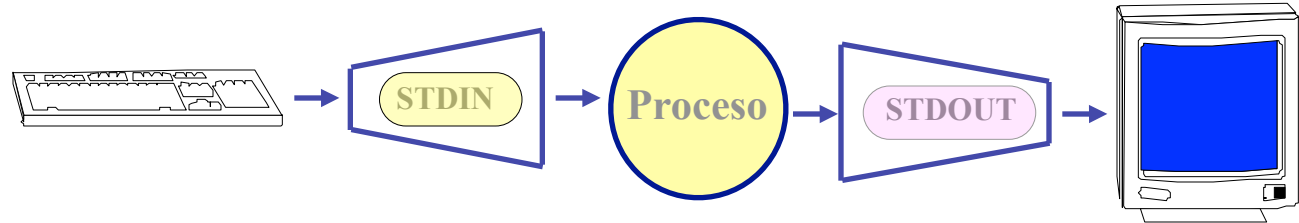
◆Clase **stream.Readable** (flujo de entrada)

- Eventos:
 - 'data' (llegada de datos), 'end' (final de flujo), 'close', ...
- Métodos:
 - setEncoding([encoding]), pause(), resume(), destroy(), ..

◆Clase **stream.Writable** (flujo de salida)

- Eventos:
 - 'pipe', 'drain', 'error', 'finish', ...
- Métodos (son bloqueantes):
 - write(string, [encoding], [fd]), write(buffer), end(), .., destroy(), ...

E/S



◆ El módulo **process** hereda los streams del S.O.

- **stdin**: entrada estándar asignada a teclado
 - stdin se arranca con “resume()”, recibe líneas tecleadas con evento “data”
 - ^D, ^C, .. cierran el flujo de entrada desde el teclado
- **stdout**: salida estándar asignada a pantalla
- **stderr**: salida de error asignada a pantalla

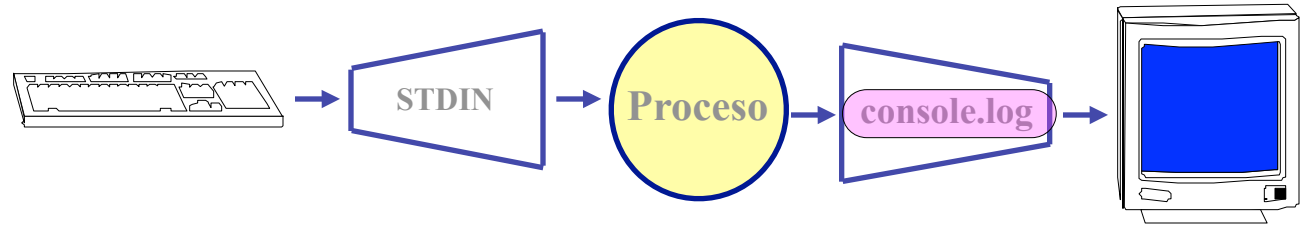
```
e90-stdin_out.js  UNREGISTERED
// arranca stream
process.stdin.resume();

// Configura entrada ASCII (sino buffer binario)
process.stdin.setEncoding('ascii');

// Manejador de evento 'data'
// -> bucle de lectura de líneas
process.stdin.on('data', function(line) {
  process.stdout.write(line);
});
```

```
t3 — bash — 28x8
venus:t3 jq$
venus:t3 jq$
venus:t3 jq$ node stdout.js
hola, hola
hola, hola
que tal
que tal
venus:t3 jq$
```

console.log



◆console.log(...)

- método de escritura en consola que formatea la salida
 - Más amigable que “process.stdout.write()”
- “process.stdout.write()” no formatea la salida

```
e91-stdin.js  UNREGISTERED
// arranca stream
process.stdin.resume();

// Configura entrada ASCII (sino buffer binario)
process.stdin.setEncoding('ascii');

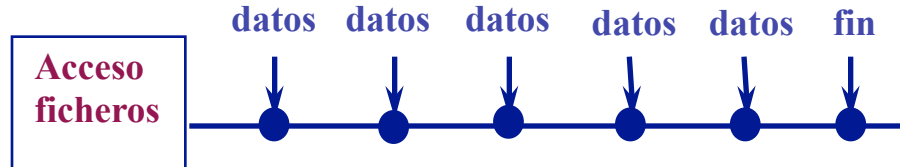
// Manejador de evento 'data'
// -> bucle de lectura de líneas
process.stdin.on('data', function(line) {
  console.log(line);
});
```

```
t3 - back - 27x9
venus:t3 jq$
venus:t3 jq$ node stdin.js
hola, hola
hola, hola

que tal
que tal

venus:t3 jq$
```

Copy con pipe



- ♦ Los streams permiten acceder a los ficheros por bloques de datos
 - El método **pipe(..)** de **fs** realiza una copia más eficaz
 - leyendo y escribiendo bloques a medida que llegan del disco
 - y no esperando a leer el fichero completo

```
var fs = require('fs');

if (process.argv.length !== 4){           // parámetros mal
  console.log('  syntax: "node copy <orig> <dest>"');
  process.exit()                          // finaliza si parámetros mal
}

//Abre ficheros como flujos (streams) de lectura y escritura
var readStream = fs.createReadStream(process.argv[2]);
var writeStream = fs.createWriteStream(process.argv[3]);

// Programa acaba al no haber eventos de copia
readStream.pipe(writeStream);
```



Final del tema