

# Sistemas Empotrados

## Tema 5: Entrada/salida

Lección 14:  
Diseño de drivers de nivel 2



# Contenidos

## Tema 5: Entrada/salida

### Introducción

El GPIO

Acceso al mapa de memoria

Diseño de drivers en capas

### Drivers de nivel 0

Introducción

Diseño de un driver L0

### Drivers de nivel 1

Introducción

Diseño de un driver L1

### Drivers de nivel 2

Introducción

Diseño de un driver L2

# Drivers de nivel 2



Acceso al dispositivo mediante llamadas a la biblioteca estándar de C (libC), proporcionando portabilidad y facilidad de uso

## Características:

- Todos los dispositivos se tratan con un API estándar (ANSI C), independiente de la plataforma
- El uso de streams (FILE \*) mejora las prestaciones de E/S
- E/S con formato (fprintf, fscanf)

## Ejemplo

```
void function (void)
{
    FILE * uart_stream;
    uart_stream = fopen("/dev/uart1/9600,n,8,1", "w");
    fprintf(uart_stream, "Uso de la uart %d mediante streams de E/S\r\n", 1);
    fclose(uart_stream);
}
```

# LibC: la biblioteca estándar de C

## ¿Qué es libC?

Proporciona las macros, tipos y funciones usadas comúnmente en el lenguaje C:

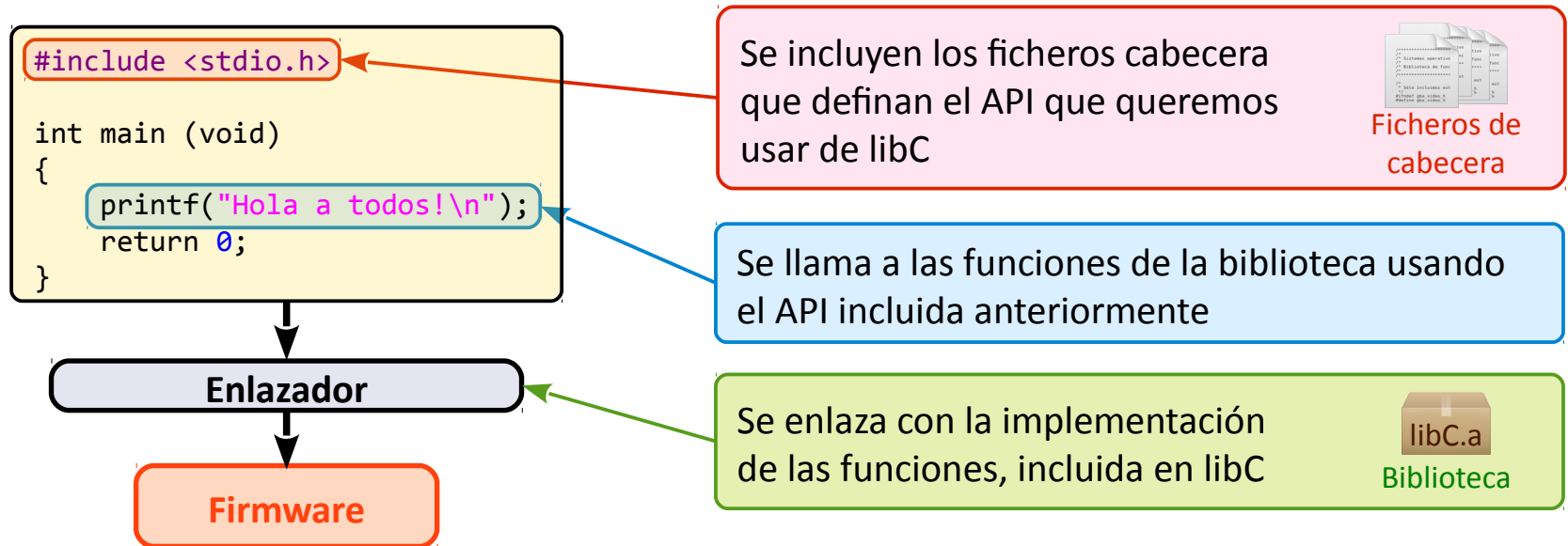
Tratamiento de cadenas: strcmp, strlen, ...

E/S: fprintf, fscanf, etc.

Gestión de procesos: fork, kill, ...

Gestión de memoria: malloc, free, ...

## ¿Cómo se usa?



Ventajas: API estándar, portabilidad entre plataformas

# E/S en la biblioteca estándar de C

## Standard I/O (stdio.h)

Estándar ANSI C (C89)

API más completa y versátil (fprintf, fscanf), basada en streams (FILE \*)

Mejora las prestaciones mediante el uso de búferes y transferencias asíncronas

Garantiza portabilidad entre diferentes plataformas

## Low level I/O (unistd.h)

Opcional, sólo si la biblioteca de C implementa el estándar POSIX

API muy sencilla (open, read, write, close), basada en file descriptors (int)

Implementación dependiente de la plataforma

Permite control total sobre los dispositivos (ioctl)

Plataforma HW

# Alternativas para libC

**glibc:** biblioteca libC de GNU, usada en casi todas las distribuciones de Linux

**Puntos fuertes:** Implementa completamente los estándares ISO C11 y POSIX.1-2008

**Punto débil:** tiene una huella de 1,2MB!

**Eglibc:** Embedded glibc, adoptada por Debian y Ubuntu

**Puntos fuertes:** compatible con glibc, mejor soporte para sistemas empujados

**Punto débil:** Sigue siendo grande (unos 900KB), no apta para sistemas con poca memoria

**uClibc:** diseñada para portar Linux a microcontroladores (uClinux)

**Puntos fuertes:** Implementación completa, menor tamaño que Eglibc (unos 420KB)

**Punto débil:** Pensada para dar soporte sólo a sistemas Linux empujados sin MMU

**dietlibc:** versión reducida de libc para enlazado estático

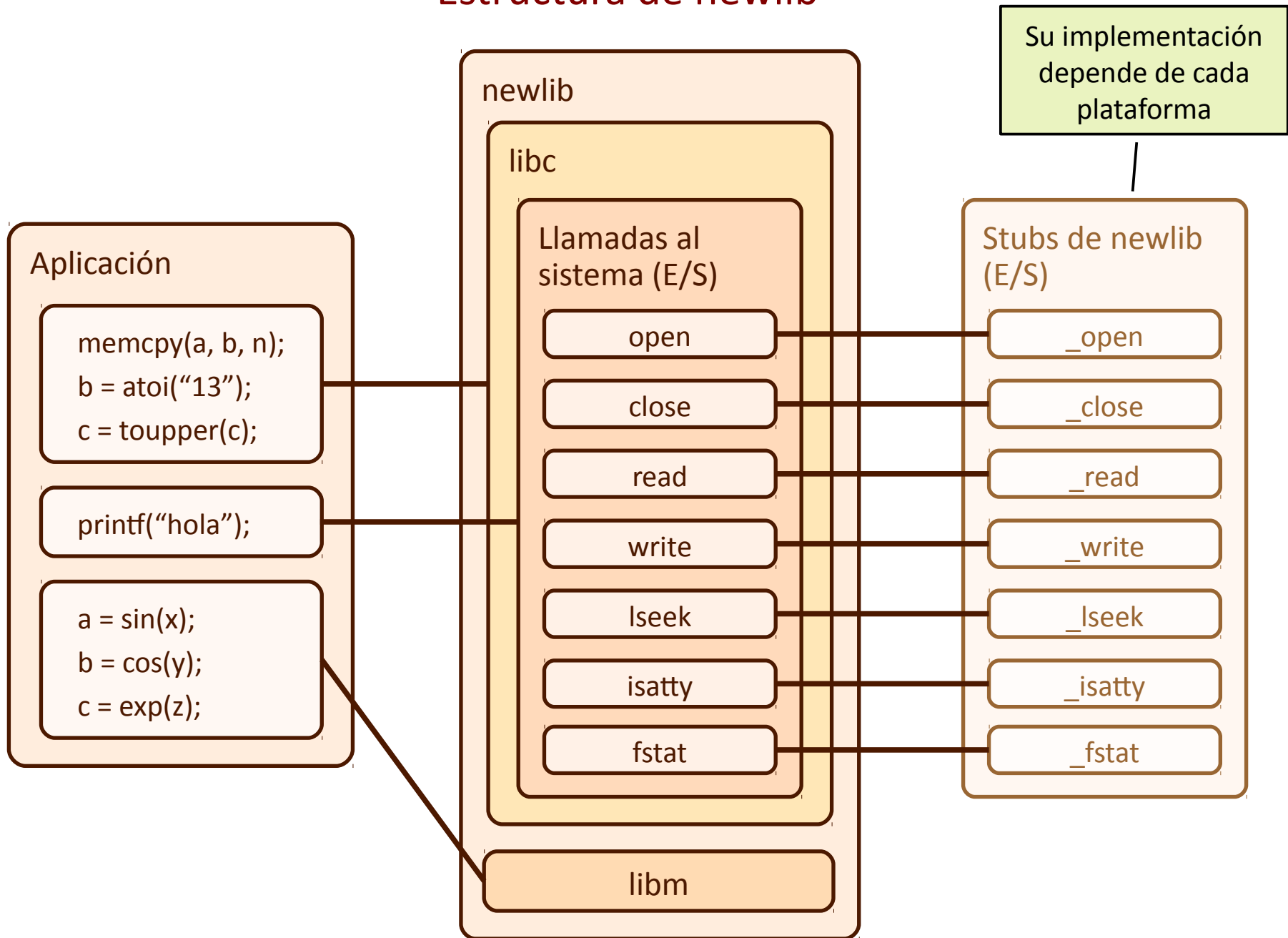
**Puntos fuertes:** diseñada para minimizar el tamaño de los ejecutables

**Punto débil:** Sólo implementa las funciones más usadas de libc

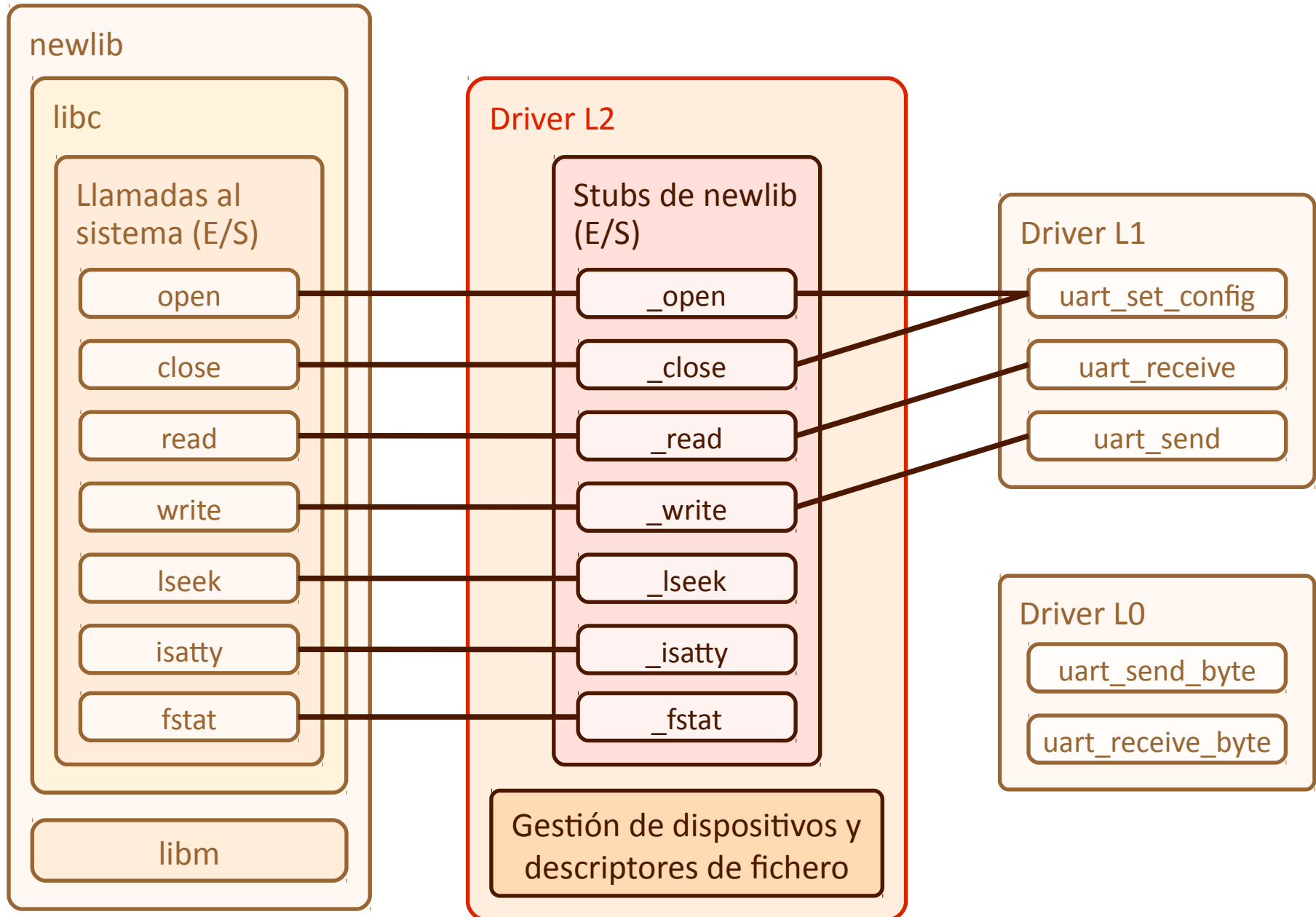
**newlib:** versión de libc orientada a sistemas empujados sin SO

**Puntos fuertes:** licencia BSD, versiones reducidas de algunas funciones, diseñada para facilitar la portabilidad. Adoptada por la mayoría de toolchains y distribuciones de GCC para sistemas empujados

# Estructura de newlib



# Integración de newlib con el driver L1





# Contenidos

## Tema 5: Entrada/salida

### Introducción

El GPIO

Acceso al mapa de memoria

Diseño de drivers en capas

### Drivers de nivel 0

Introducción

Diseño de un driver L0

### Drivers de nivel 1

Introducción

Diseño de un driver L1

### Drivers de nivel 2

Introducción

Diseño de un driver L2

# Ejemplo de uso

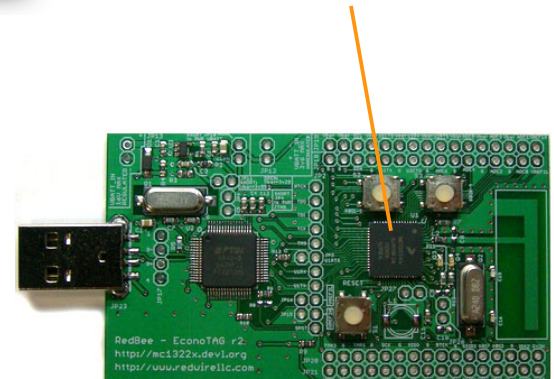
```
picocom
picocom v1.4

port is      : /dev/ttyUSB1
flowcontrol  : none
baudrate is  : 115200
parity is    : none
databits are : 8
escape is    : C-a
noinit is    : no
noreset is   : no
nolock is    : no
send_cmd is  : ascii_xfr -s -v -l10
receive_cmd is : rz -vv

Terminal ready
Hola mundo!
```

```
#include <stdio.h>
#include "system.h"

int main (void)
{
    printf("Hola mundo!\n");
    return 0;
}
```



# La lista de dispositivos

## bsp/hal/include/dev.h

```
/** Estructura para almacenar las funciones de gestión de cada dispositivo */
typedef struct
{
    const char  *name;                /* Nombre del dispositivo */
    uint32_t id;                      /* Identificador. Por defecto es cero. Se usa para */
                                     /* diferenciar a varios dispositivos del mismo tipo */

    int (*open)(uint32_t id, int flags, mode_t mode); /* Función open */
    int (*close)(uint32_t id);         /* Función close */
    ssize_t (*read)(uint32_t id, char *buf, size_t count); /* Función read */
    ssize_t (*write)(uint32_t id, char *buf, size_t count); /* Función write */
    off_t (*lseek)(uint32_t id, off_t offset, int whence); /* Función lseek */
    int (*fstat)(uint32_t id, struct stat *buf); /* Función fstat */
    int (*isatty)(uint32_t id);        /* Función isatty */
} bsp_dev_t;
```

Asocia cada dispositivo con las funciones que implementan su interfaz POSIX

## bsp/hal/dev.c

```
/** Lista de dispositivos registrados */
static bsp_dev_t bsp_dev_list [BSP_MAX_DEV] =
{
    {
        "/dev/null", /* La primera entrada de la tabla es /dev/null */
        0,           /* Identificador */
        NULL,        /* Función open por defecto */
        NULL,        /* Función close por defecto */
        NULL,        /* Función read por defecto */
        NULL,        /* Función write por defecto */
        NULL,        /* Función lseek por defecto */
        NULL,        /* Función fstat por defecto */
        NULL,        /* Función isatty por defecto */
    }
    /* El resto del array se inicializa a cero */
};
```

Lista de dispositivos gestionados por el BSP

Implementada como una tabla por cuestiones de eficiencia y para evitar el uso de memoria dinámica por seguridad

# Registro de dispositivos en el BSP

## bsp/hal/include/dev.h

```
/**
 * Registro de un dispositivo en el sistema.
 * @param name      Nombre del dispositivo
 * @param id         Identificador del dispositivo
 * @param open       Función open del dispositivo
 * @param close      Función close del dispositivo
 * @param read       Función read del dispositivo
 * @param write      Función write del dispositivo
 * @param lseek      Función lseek del dispositivo
 * @param fstat      Función fsat del dispositivo
 * @param isatty     Función isatty del dispositivo
 * @return           El numero de dispositivo asignado o -1 en caso de error
 */
int32_t bsp_register_dev (const char *name, uint32_t id, int (*open)(uint32_t id, int flags, mode_t mode),
                          int (*close)(uint32_t id), ssize_t (*read)(uint32_t id, char *buf, size_t count),
                          ssize_t (*write)(uint32_t id, char *buf, size_t count), off_t (*lseek)(uint32_t id, off_t offset, int whence),
                          int (*fstat)(uint32_t id, struct stat *buf), int (*isatty)(uint32_t id));
```

Insertamos un dispositivo en la tabla de dispositivos, asociándole las funciones que implementan su interfaz POSIX

## bsp/drivers/uart.c

```
/**
 * Inicializa una uart. Implementación de nivel 2
 * @param uart      Identificador de la uart
 * @param br        Baudrate
 * @param name      Nombre del dispositivo
 * @return          Cero en caso de éxito o -1 en caso de error. La condición de error se indica en la variable global errno
 */
```

Al inicializar un dispositivo lo registramos en la lista de dispositivos del sistema

```
int32_t uart_init (uart_id_t uart, uint32_t br, const char *name)
```

```
{
```

### Implementación L1

```
/* Registramos el dispositivo. Implementación del driver de nivel 2 */
bsp_register_dev (name, uart, NULL, NULL, uart_receive, uart_send, NULL, NULL, NULL);
return 0;
}
```

# La lista de ficheros abiertos

## Motivación

La interfaz POSIX (acceso a bajo nivel) de los dispositivos está basada en **descriptores de fichero**, que serán entradas de la tabla de ficheros abiertos del BSP

Puede haber más de un fichero abierto sobre el mismo dispositivo

**STDOUT, STDIN, STDERR** sobre **/dev/uart1** (es lo más habitual)

### bsp/hal/include/dev.h

```
/** Estructura minimalista de un descriptor de fichero */
typedef struct
{
    bsp_dev_t* dev;          /* Puntero a la estructura gestión del dispositivo */
    int        flags;        /* Flags de apertura/creación del fichero */
} bsp_fd_t;
```

De momento, no pretendemos soportar un sistema de archivos completo en nuestro BSP

### bsp/hal/dev.c

```
/**
 * Lista de descriptores de ficheros abiertos. Las primeras tres entradas
 * se reservan para E/S estándar, asignada por defecto a /dev/null.
 */
static bsp_fd_t bsp_fd_list [BSP_MAX_FD] =
{
    { bsp_dev_list, 0 }, /* Entrada estándar (STDIN) -> /dev/null */
    { bsp_dev_list, 0 }, /* Salida estándar (STDOUT) -> /dev/null */
    { bsp_dev_list, 0 }, /* Error estándar (STDERR) -> /dev/null */
    /* El resto de entradas se inicializan a cero */
};
```

Una vez inicializados los dispositivos, se redireccionarán estos tres descriptores de fichero al dispositivo usado por el BSP para la E/S estándar

# Redirección de la E/S estándar

bsp/hal/include/dev.h

```
/**
 * Abre un dispositivo y lo asigna al descriptor de fichero especificado,
 * en vez de crear una nueva entrada en la tabla de descriptors de fichero.
 * @param fd    Descriptor de fichero al que se redireccionará el dispositivo
 * @param name  Nombre del dispositivo
 * @param flags Configuración
 * @param mode  Modo de apertura
 */
void redirect_fd (uint32_t fd, const char* name, int flags, mode_t mode);
```

Función necesaria para poder redireccionar la E/S estándar al dispositivo que nos interese

bsp/hal/bsp\_init.c

```
/**
 * Redirección de la E/S estándar.
 * Esta función se llama por el BSP antes de llamar a main, y una vez que se
 * han inicializado los dispositivos, para fijar la E/S estándar del sistema,
 * que por defecto está asignada a /dev/null.
 * @param stdin_dev  Nombre del dispositivo que se usará para la entrada estándar
 * @param stdout_dev Nombre del dispositivo que se usará para la salida estándar
 * @param stderr_dev Nombre del dispositivo que se usará para el error estándar
 */
static void bsp_io_redirect (const char* stdin_dev, const char* stdout_dev, const char* stderr_dev)
{
    /* Redireccionamos la E/S estándar */
    redirect_fd (STDIN_FILENO, stdin_dev, O_RDONLY, 0777);
    redirect_fd (STDOUT_FILENO, stdout_dev, O_WRONLY, 0777);
    redirect_fd (STDERR_FILENO, stderr_dev, O_WRONLY, 0777);
}
```

Una vez inicializados los dispositivos, se puede usar cualquier dispositivo de caracteres para implementar la E/S estándar

Definidos como 0, 1  
y 2 en <unistd.h>

Definidos en  
<fcntl.h>

# Ejemplo de inicialización del sistema

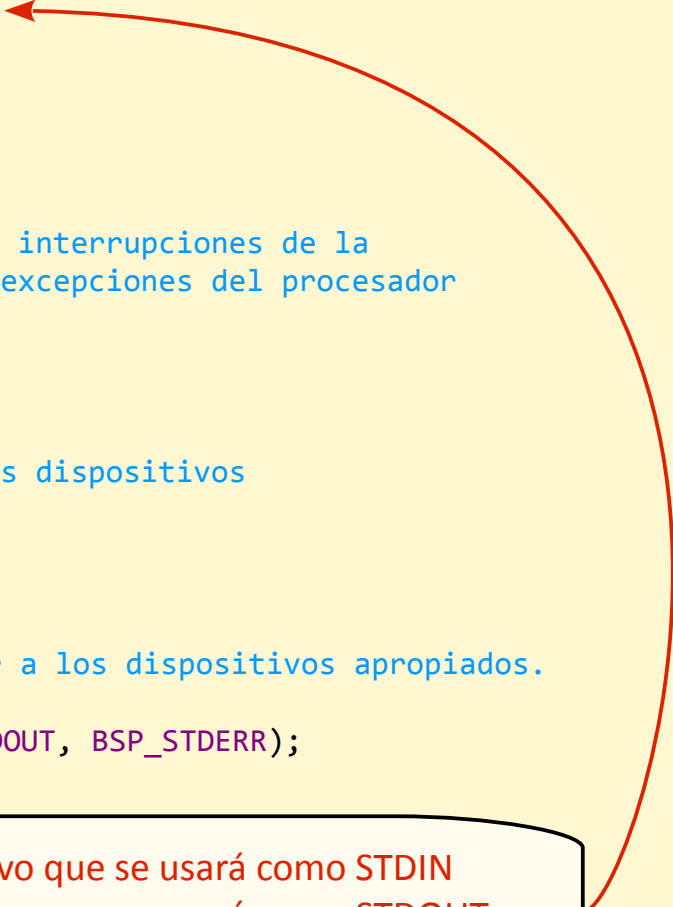
## Inicialización del BSP

```
/*
 * Configuración de E/S estándar
 */
#define BSP_STDOUT    "/dev/uart1"
#define BSP_STDIN     "/dev/uart1"
#define BSP_STDERR    "/dev/uart1"

void bsp_init (void)
{
    /*
     * Inicializamos el controlador de interrupciones de la
     * plataforma y la gestión de las excepciones del procesador
     */
    bsp_excep_init();

    /*
     * Inicializamos los drivers de los dispositivos
     */
    bsp_sys_init();

    /*
     * Redireccionamos la E/S estándar a los dispositivos apropiados.
     */
    bsp_io_redirect(BSP_STDIN, BSP_STDOUT, BSP_STDERR);
}
```



BSP\_STDIN → Dispositivo que se usará como STDIN  
BSP\_STDOUT → Dispositivo que se usará como STDOUT  
BSP\_STDERR → Dispositivo que se usará como STDERR

# Implementación de \_open

bsp/hal/syscalls.c

```
/**
 * Abre un dispositivo/fichero
 * @param pathname Nombre del dispositivo/fichero
 * @param flags     Modo de acceso
 * @param mode      Permisos en caso de que se cree un archivo/dispositivo
 * @return          El descriptor del dispositivo o -1 en caso de error.
 *                 La condición de error se indica en la variable global errno.
 */
int _open (const char *pathname, int flags, mode_t mode)
{
    bsp_dev_t *dev = find_dev (pathname);    /* Buscamos el dispositivo en la tabla de dispositivos del BSP */

    if (dev) /* Si existe el dispositivo */
    {
        if (dev->open) /* y si el dispositivo tiene implementada la función open */
            if (dev->open(dev->id, flags, mode) >= 0) /* y si no falla la llamada a open, se retorna su descriptor */
                return get_fd(dev, flags);
            else
                errno = ENOTSUP; /* Operación no soportada */
    }
    else
        errno = ENODEV; /* El dispositivo no existe */

    return -1;
}
```

Por defecto se puede abrir cualquier dispositivo que exista en el sistema

Más información: man 2 open

Caso			Retorno	errno
El dispositivo no existe			-1	ENODEV
El dispositivo existe	No tiene función open			ENOTSUP
	Tiene función open	La función open falla		Fijado por dev->open
		La función open no falla	fd	



# Implementación de \_close

bsp/hal/syscalls.c

```
/**
 * Cierra un dispositivo/fichero
 * @param fd Descriptor de fichero/dispositivo
 * @return 0 en caso de éxito o -1 en caso de error.
 * La condición de error se indica en la variable global errno.
 */
int _close (int fd)
{
    bsp_dev_t *dev = get_dev(fd);

    release_fd (fd); /* Liberamos el descriptor de fichero */

    if (dev && dev->close)
        return dev->close(dev->id);
    else
    {
        /* fd no es un descriptor de fichero abierto válido */
        errno = EBADF;
        return -1;
    }
}
```

Por defecto no se puede cerrar  
ningún dispositivo, salvo que  
tenga definida una función close

Más información: man 2 close

Caso			Retorno	errno
El dispositivo no existe			-1	EBADF
El dispositivo existe	No tiene función close			Fijado por dev->close
	Tiene función close	La función close falla	0	
		La función close no falla		

# Implementación de \_read

bsp/hal/syscalls.c

```
/**
 * Lectura de un dispositivo/fichero
 * @param fd    Descriptor de fichero/dispositivo
 * @param buf    Puntero al búfer donde se almacenarán los datos
 * @param count  Número de bytes que se quieren leer
 * @return       El número de bytes leídos o -1 en caso de error.
 *              La condición de error se indica en la variable global errno.
 */
ssize_t _read (int fd, char *buf, size_t count)
{
    bsp_dev_t *dev = get_dev(fd);

    if (dev && dev->read)
        return dev->read(dev->id, buf, count);
    else
        return 0;
}
```

Por defecto no se puede leer  
de ningún dispositivo si no  
tiene definida una función read

Más información: [man 2 read](#)

Caso			Retorno	errno
El dispositivo no existe			0	
El dispositivo existe	No tiene función read			
	Tiene función read	La función read no falla	# bytes leídos	
		La función read falla	-1	Fijado por dev->read

# Implementación de `_write`

bsp/hal/syscalls.c

```
/**
 * Escritura en un dispositivo/fichero
 * @param fd    Descriptor de fichero/dispositivo
 * @param buf    Puntero al búfer que almacena los datos
 * @param count  Número de bytes que se quieren escribir
 * @return       El número de bytes escritos o -1 en caso de error.
 *              La condición de error se indica en la variable global errno.
 */
ssize_t _write (int fd, char *buf, size_t count)
{
    bsp_dev_t *dev = get_dev(fd);

    if (dev && dev->write)
        return dev->write(dev->id, buf, count);
    else
        return count;
}
```

Por defecto se supone que se han escrito todos los bytes, salvo que la función `write` del dispositivo no pueda hacerlo

Más información: `man 2 write`

Caso

Retorno

errno

El dispositivo no existe

# bytes a escribir

El dispositivo existe

No tiene función `write`

Tiene función `write`

La función `write` no falla

La función `write` falla

# bytes escritos

-1

Fijado por `dev->write`

# Implementación de \_lseek

bsp/hal/syscalls.c

```
/**
 * Modificación del desplazamiento en un dispositivo/fichero
 * @param fd      Descriptor de fichero/dispositivo
 * @param offset  Desplazamiento
 * @param whence  Base para el desplazamiento
 * @return        El nuevo desplazamiento medido en bytes desde el inicio del fichero o -1 en caso de error.
 *               La condición de error se indica en la variable global errno.
 */
off_t _lseek (int fd, off_t offset, int whence)
{
    bsp_dev_t *dev = get_dev(fd);

    if (dev && dev->lseek)
        return dev->lseek(dev->id, offset, whence);
    else
        return 0;
}
```

Por defecto se puede hacer el desplazamiento

Más información: [man 2 lseek](#)

Caso

Retorno

errno

El dispositivo no existe

0

El dispositivo existe

No tiene función lseek

Tiene función lseek

La función lseek no falla

La función lseek falla

# bytes desplz.

-1

Fijado por dev->lseek

# Implementación de \_fstat

bsp/hal/syscalls.c

```
/**
 * Obtención de información de un dispositivo/fichero
 * @param fd  Descriptor de fichero/dispositivo
 * @param buf  Estructura para almacenar dicha información
 * @return    0 o -1 en caso de error.
 *           La condición de error se indica en la variable global errno
 */
int _fstat (int fd, struct stat *buf)
{
    bsp_dev_t *dev = get_dev(fd);

    if (dev && dev->fstat)
        return dev->fstat(dev->id, buf);
    else
    {
        buf->st_mode = S_IFCHR; /* Dispositivo de caracteres */
        return 0;
    }
}
```

Por defecto se asume que el dispositivo es de caracteres

Más información: [man 2 fstat](#)

Caso		Retorno	errno
El dispositivo no existe		0	
El dispositivo existe	No tiene función fstat		
	Tiene función fstat		
	La función fstat no falla	-1	Fijado por dev->fstat
	La función fstat falla		

# Implementación de \_isatty

bsp/hal/syscalls.c

```
/**
 * Chequea si el descriptor de fichero corresponde con una terminal
 * @param fd Descriptor de fichero/dispositivo
 * @return 1 en caso de que el descriptor corresponda a una terminal o 0 en otro caso.
 * La condición de error se indica en la variable global errno
 */
int _isatty (int fd)
{
    bsp_dev_t *dev = get_dev(fd);

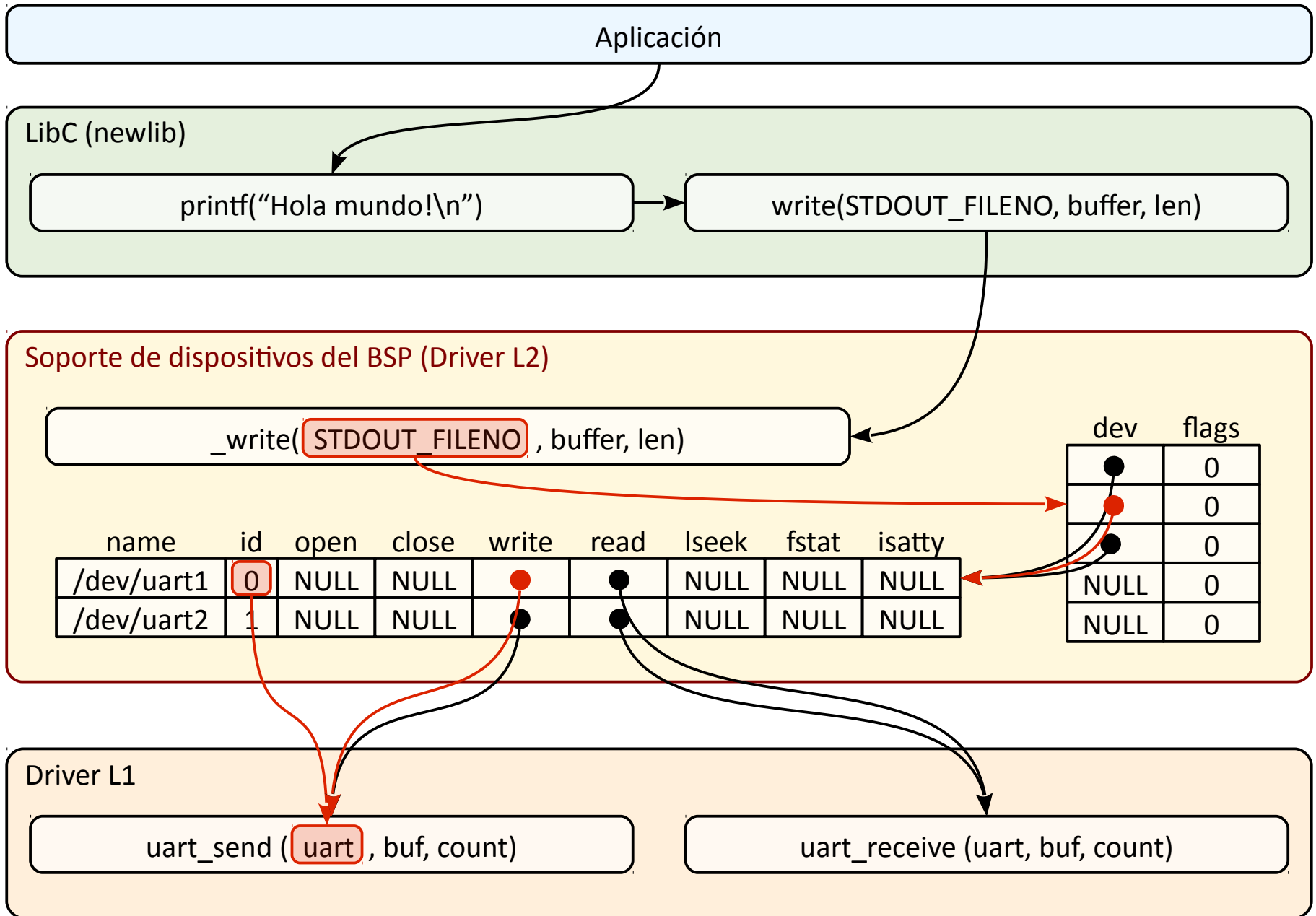
    if (dev && dev->isatty)
        return dev->isatty(dev->id);
    else
        return 1; /* Es una terminal */
}
```

Por defecto se asume que el dispositivo es una terminal

Más información: `man isatty`

Caso			Retorno	errno
El dispositivo no existe			1	
El dispositivo existe	No tiene función isatty			
	Tiene función isatty	La función isatty no falla		
		La función isatty falla	0	Fijado por dev->isatty

# Visión global



# Lecturas recomendadas

B. Gatliff. *Porting and Using Newlib in Embedded Systems*. BillGatliff.com.

<http://neptune.billgatliff.com/newlib.html>

Rob Savoye. *Embed with GNU. Porting the GNU Tools to Embedded Systems*. Cygnus, 1995.

<http://www.gnuarm.com/pdf/porting.pdf>

J. Bennett. *Howto: Porting Newlib. A Simple Guide*, 2010. Capítulo 5.2

<http://www.embecosm.com/download/ean9.html>

S. Chamberlain, R. Pesch, Red Hat Support y J. Johnston. *Red Hat Newlib C Library: Full Configuration*, online. <http://sourceware.org/newlib/libc.html>

Francesco Balducci. *Using Newlib in ARM bare metal programs*. Balau Blog, 2010.

<http://balau82.wordpress.com/2010/12/16/using-newlib-in-arm-bare-metal-programs/>