Uso de printf() y scanf() en el microcontrolador NXP LPC4337

Biblioteca newlib, printf() y scanf() completas o reducidas

La biblioteca estándar de C llamada **newlib,** que incluye el compilador **arm-none-eabi** del **NXP LPC4337** (el microcontrolador que tiene al EDU-CIAA-NXP o la CIAA-NXP) permite utilizar printf() y scanf() con el microcontrolador.

La biblioteca newlib permite ser compilada en 2 versiones. Una versión completa y una versión reducida nombrada newlib nano.

Si se utiliza newlib nano printf() y scanf() no permiten imprimir o recibir valores en punto flotante (float).

Para desactivar newlib nano y que compila la new lib completa en la carpeta app del proyecto, el archivo config.mk poner:

USE_NANO=n

Por defecto viene en y (que si use nano).

De esta manera podemos poner en un programa:

```
printf( "Pi vale aproximadamente %f\n", 3.14 );
```

scanf() y printf() utilizan la UART2 del microcontrolador (UART_USB en la sAPI) como entrada/salida estándar por lo que no se debería reconfigurar la UART2, la está configurada por defecto a 115200 baudios. Esto se hace en el *startup* del microcontroaldor.

Entonces un programa básico que utilice printf() puede ser simplemente:

```
#include "sapi.h" // <= Biblioteca sAPI, o alternativamente puede
                  // usarse board.h que incluye LPCOpen
int main( void )
{
   printf( "Pi vale aproximadamente %f\n", 3.14 );
   while( TRUE );
   return 0;
}
```

NOTA:

Si usan printf() en un microcontrolador van a notar que no imprime directamente (envía directo a la UART) si no que lo hace a través de un buffer el cual se envía a la salida estándar, en nuestro caso la UART.

En un programa en la PC si no hay \n este buffer se envía a la salida estándar cuando se llama a la función fflush(), la cual por ejemplo se llama cuando termina el programa.

En embebidos el programa no termina nunca (por el while(1)), entonces hay que forzar el fflush de 2 formas:

1. Llamando a la función fflush():

```
fflush(stdout);
```

2. Con \n dentro del texto de printf()

```
printf( "Hola\n" );
```

Uso de memorias del microcontrolador

Se puede probar compilar el mismo programa con la opción USE_NANO en "y" y también en "n" y observar la diferencia de ocupación de memorias.

Con newlib nano

```
   text    data     bss     dec     hex filename
  11460     300      76   11836    2e3c app/out/app.elf
```

En este caso printf() no imprime los valores float %f, pero no da errores, simplemente los ignora.

Con newlib completa

```
   text    data     bss     dec     hex filename
  28992    2676     152   31820    7c4c app/out/app.elf
```

En este caso printf() funciona con todas sus características.

Para comprender esos números, todos están en **bytes** y representan:

- **text** - contiene el código compilado y datos de sólo lectura (read-only) de la aplicación (el valor de bytes está en decimal).
- **data** - shows the read-write data in your application (el valor de bytes está en decimal).
- **bss** - Muestra los datos inicializados con 0 ('bss' y 'common') en la aplicación (el valor de bytes está en decimal).
- **dec** - Total = 'text' + 'data' + 'bss' (el valor de bytes está en decimal).
- **hex** - El valor de bytes en hexadecimal equivalente al de 'dec'.

Esto típicamente se interpreta como:

- Consumo de **FLASH** de la aplicación equivale a **text + data**.
- Consumo de **RAM** de la aplicación equivale a **data + bss**.

Más información en: http://www.support.code-red-tech.com/CodeRedWiki/FlashRamSize

Como puede observarse, newlib completa ocupa muchos más recursos.

Uso de printf() y scanf()

printf()

Las funciones de la familia printf producen una salida de acuerdo a *format* como se describe abajo. Printf y vprintf escriben su salida a *stdout*, el flujo de salida estándar. fprintf y vfprintf escriben su salida al *stream* de salida dado. sprintf, snprintf, vsprintf y vsnprintf escriben a una cadena de caracteres *str*.

Prototipo o declaración

```
int printf( const char *format, ... );
```

Ejemplos

```
printf("Strings:\n");
const char* s = "Hello";
printf("\t.%10s.\n\t.%-10s.\n\t.%*s.\n", s, s, 10, s);

printf("Characters:\t%c %%\n", 65);
```

```
printf("Integers\n");
printf("Decimal:\t%i %d %.6i %i %.0i %+i %u\n",1,2,3,0,0,4,-1);
printf("Hexadecimal:\t%x %x %X %#x\n", 5, 10, 10, 6);
printf("Octal:\t%o %#o %#o\n", 10, 10, 4);

printf("Floating point\n");
printf("Rounding:\t%f %.0f %.32f\n", 1.5, 1.5, 1.3);
printf("Padding:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);
printf("Scientific:\t%E %e\n", 1.5, 1.5);
printf("Hexadecimal:\t%a %A\n", 1.5, 1.5);
```

Los caracteres % (especificaciones de conversión)

Conversion Character	***What It Displays***
%%	Percent character (%)
%c	Single character (char)
%d	Integer value (short, int)
%e	Floating-point value in scientific notation using a little E (float, double)
%E	Floating-point value in scientific notation using a big E (float, double)
%f	Floating-point value in decimal notation (float, double)
%g	Substitution of `%f` or `%e`, whichever is shorter (float, double)
%G	Substitution of `%f` or `%E`, whichever is shorter (float, double)
%i	Integer value (short, int)
%ld	Long integer value (long int)
%o	Unsigned octal value; no leading zero
%p	Memory location in hexadecimal (*pointer)
%s	String (char *)
%u	Unsigned integer (unsigned short, unsigned int, unsigned long)
%x	Unsigned hexadecimal value, lowercase (short, int, long)
%X	Unsigned hexadecimal value, capital letters (short, int long)

Más información

- http://es.tldp.org/Paginas-manual/man-pages-es-1.28/man3/printf.3.html
- https://es.cppreference.com/w/c/io/fprintf

scanf()

La familia **scanf** de funciones busca en la entrada según un *formato* como se describe más adelante. Este formato puede contener *especificadores de conversión*; los resultados de tales conversiones, si las hay, se guardan donde apunten los argumentos *punteros*. La función **scanf** lee la entrada del flujo de entrada estándar *stdin*, **fscanf** lee su entrada del puntero a FILE *flujo*, y **sscanf** lee su entrada de la cadena de caracteres a la que apunte *str*.

Prototipo o declaración

```
int scanf( const char *format, ... );
```

Ejemplo

```
char text[50];
printf( "Ingrese un texto y presione enter:\r\n" );
scanf( "%s", text );
printf( "El texto ingresado es: %s\r\n", text );
```

Los carcateres % (especificaciones de conversión)

%c para indicar entrada de un caracter
%d para indicar entrada de un entero
%f para indicar entrada de un flotante
%s para indicar entrada de un string

Sugerencias para el uso de scanf()

- Antes de usar scanf, imprimir con printf() indicando al usuario qué tipo de dato o información debe ingresar.
- Ingresar de a un dato por vez. Será mejor que cada llamado a scanf() sea para ingresar un dato, evitar el ingreso múltiple.
- Para ingresar texto, indicar el máximo de caracteres que scanf() debe aceptar, así no almacena en zonas inapropiadas.

Más información

- http://es.tldp.org/Paginas-manual/man-pages-es-1.28/man3/scanf.3.html
- https://es.cppreference.com/w/c/io/fscanf

Ejemplo con scanf() y printf()

```
#include "sapi.h" // <= Biblioteca sAPI, o alternativamente puede
                  // usarse board.h que incluye LPCOpen

// La UART_USB se inicializa en el startup del microcontrolador a
// 115200 baudios  para salida y entrada estandar.

// Recordar en config.mk poner USE_NANO=n (usar newlib completa
// en lugar de newlib nano) para tener soporte de flotantes en
// printf(), sin embargo, esto causa que el programa ocupe MUCHA
// mas RAM y FLASH.

int main( void )
{
   printf( "Pi vale aproximadamente %f\r\n", 3.14 );

   char text[50];
   int num;

   printf( "Ingrese un texto y presione enter:\r\n" );
   scanf( "%s", text );
   printf( "El texto ingresado es: %s\r\n", text );

   printf( "Ingrese un numero y presione enter:\r\n" );
   scanf( "%d", &num );
   printf( "El numero ingresado es: %d\r\n", num );

   while( TRUE );

   return 0;
}
```