4.3.1

La función para la recepción de datos es: unsigned int uart\_getc(void);

Como se puede ver es una función sin parámetros en la que se devuelve un entero sin signo, 2 bytes, en los que el byte menos significativo hace referencia al carácter recibido y el más significativo es utilizado para el control de errores.

Ejemplo:

char c = uart\_getc( );

while (c != ‘\0’){ … }

4.3.2

Las funciones para el envio de datos son:

1. void uart\_putc(unsigned char data);

Esta función tiene un parámetro, el dato que quiere ser transmitido mediante UART. Dicho dato es de tipo unsigned char (1 byte).

Ejemplo: uart\_putc('\r');

1. void uart\_puts(const char \*s );

En este caso también encontramos un sólo parámetro, una cadena que será transmitida mediante UART caracter a caracter.

Ejemplo: uart\_puts("String stored in SRAM\n");

1. void uart\_puts\_p(const char \*s );

Este caso es igual al anterior, salvo porque la cadena proviene de la memoria del programa. Esta función hace uso de la macro uart\_puts\_P.

Ejemplo: uart\_puts\_P("String stored in FLASH\n");

4.3.3

En primer lugar, es necesario inicializar el puerto USB y configurarlo. Dicha inicialización implica, entre otras tareas y por citar un ejemplo, la especificación de la velocidad a la que va a trabajar con el puerto (9600 baudios). Para poder llevar a cabo dicha inicialización será necesario incluir en nuestro programa las bibliotecas de gestión y control de E/S. En nuestro caso utilizaremos para inicializar el puerto la función contenida en la biblioteca ticcommpc.cpp: InicializarUSB(USBPORT). Para llevar a cabo la compilación tendremos por un lado el programa para PC y por otro, una biblioteca con funciones para la gestión de las comunicaciones serie. Uniendo ambas obtendremos un ejecutable para un PC.

4.3.4

En el caso de Arduino Uno las acciones son prácticamente las mismas que para PC, destacando que aparte de todo lo anterior también será necesario incluir la biblioteca para la gestión de interrupciones hardware. Con respecto a la compilación tendremos por un lado el programa para Arduino Uno y por otro, una biblioteca con funciones para la gestión de las comunicaciones serie de manera análoga a como se haría en PC (ticcommardu.cpp). Uniendo ambas obtendremos un ejecutable para un Arduino Uno.

4.3.5

Básicamente, la utilización de dicha función consiste en pasar como parámetro el nombre del fichero del dispositivo que queremos usar, en este caso el del puerto USB. Un ejemplo sería:

fd = InicializarUSB("/dev/ttyACM0");

cout << “El puerto COM es: “ << fd;

CerrarUSB(fd);

En fd, descriptor de fichero, tendremos el resultado que nos devuelva la función InicializarUSB, siendo este -1 si el puerto no ha sido asignado y por lo tanto ha habido un error en la inicialización.

A grandes rasgos y sin entrar en demasiados detalles, las acciones que realiza dicha función de manera interna son:

- Abrir el puerto USB en modo lectura y escritura y sin bloqueo.

- Configurar la velocidad de entrada y salida, en correspondencia con la asignada a Arduino.

- Personalizar las opciones del puerto.

- Devolver el descriptor de fichero.

4.3.6

Los códigos de error son los siguientes:

1. UART\_FRAME\_ERROR – Cuando se produce un error en la recepción del paquete de datos.
2. UART\_OVERRUN\_ERROR – Cuando se recibe un dato antes de leer el que había en el puerto UART porque las interrupciones hardware está configuradas lenta.
3. UART\_PARITY\_ERROR – Cuando se provoca una falta de coincidencia de paridad en los datos recibidos.
4. UART\_BUFFER\_OVERFLOW – Cuando buffer del puerto UART está lleno porque nos envían los datos a más velocidad de la que podemos leer.
5. UART\_NO\_DATA – Cuando no hay datos en el puerto UART

4.3.7

ssize\_t read(int *fildes*, void \**buf*, size\_t *nbyte*);

La función read intenta leer “nbyte” bytes del fichero asociado al descriptor de fichero abierto “fildes” en el buffer apuntado por “buf”. Si “nbyte” es 0, read devuelve 0. En caso de ocurrir algún error se devuelve -1 y se establece errno para indicar el error. Por último, en caso de que todo transcurra de manera satisfactoria se devolverá un numero entero no negativo indicando el número de bytes leídos.

ssize\_t write(int *fildes*, const void \**buf*, size\_t *nbyte*);

La función write intenta escribir “nbyte” bytes del buffer apuntado por “buf” en el fichero asociado al descriptor de fichero abierto “fildes. La devolución de esta función es idéntica a la anterior, salvando que en lugar de devolver el número de bytes leídos devolverá el número de bytes escritos.

4.3.8

El efecto de la función tcflush es el de vaciar los datos escritos pero no transmitidos o recibidos pero no leídos en el objeto fd (descriptor de fichero) pasado como parámetro a la función. En el ejemplo propuesto, TCIFLUSH, lo que haría sería borrar la cola de entrada que contienen los datos recibidos pero no leídos aún. En el caso de no usarse dicha sentencia los caracteres no leídos se acumularían en la cola de entrada si no diera tiempo suficiente a ser leídos, por lo que siguientes lecturas serían erróneas. Para nosotros en la práctica no ocurrirá nada puesto que no vamos a trabajar a distintas velocidades.