Programación de puerto serie en GNU/Linux
– Prácticas con Arduino –

Gonzalo F. Perez Paina

24 de octubre de 2018

Índice

1. Objetivos

Este documento tiene como objetivo brindar una introducción a la programación de la comunicación serial asíncrona (UART) en la PC utilizando el lenguaje de programación C en sistemas operativos GNU/Linux. La comunicación serie sirve de interfaz entre la PC y dispositivos externos, tales como placas basadas en microcontroladores, instrumentos electrónicos, otras PC, etc.

Se comienza con una breve descripción de los diferentes dispositivos de hardware que permiten obtener un puerto de comunicación serie a partir del puerto USB. Luego, se describe cómo acceder a dichos dispositivos desde el sistema operativo GNU/Linux, para finalmente mostrar su programación en lenguaje C a través del manejo de archivos de dispositivos. Para esto último se hace un brever repaso de manejo de archivos utilizando la biblioteca estándar del lenguaje C.

2. Introducción a las interfaces de comunicación serial

Previo a que el USB (Universal Serial Bus) se convirtiera en un estándar como puerto de comunicación de las PC, existían diferentes puertos tales como el PS2, utilizado particularmente para teclado y mouse, y los antiguos puertos paralelos y seriales, utilizados en sus orígenes para la comunicación con impresoras y modems, respectivamente. El puerto paralelo y el serie eran adecuados para desarrollar circuitos electrónicos propios que tuvieran la capacidad de comunicarse con la PC. La diferencia fundamental entre ellos es el tipo de comunicación “paralela vs. serial”, en las cuales para el primer caso

los bits de datos se envían en conjunto (4 u 8 bits) mientras que para el segundo los bits son enviados en serie de forma secuencial. Ambos puertos de comunicación estaban definidos mediante estándares, el IEEE-1284[1] para el puerto paralelo y el EIA/RS-232[2] para el serie. El estándar RS232 define los valores eléctricos, tales como los niveles de tensión, capacidad de corriente, impedancia, etc.

Aún cuando es común hacer referencia a "puerto serie" como equivalente al estándar RS232, este no es el único puerto serial usado para comunicar dispositivos electrónicos, existen otros tales como SPI (Serial Peripheral Interface), I^2C (Inter-Integrated Circuit), el ya mencionado USB, etc. Además, la comunicación serial puede ser de forma síncronas o asíncronas.

El circuito digital encargado de realizar la comunicación serial más comúnmente empleada en desarrollos electrónicos se conoce como UARTs (Universal Asynchronous Receiver-Transmitter) o receptor/transmisor asíncrono universal, que implementa la trama de comunicación serial estándar pero con niveles de tensiones que pueden ser de 5V o 3.3V (lo que también se conoce como serie-TTL). Esto hace que sea necesario utilizar un circuito adicional (transeiver) para adaptar las características eléctricas a las definidas por el estándar RS232. La tabla 1 muestra el nombre de cada señal del conector DB9 y la dirección de la misma. Cabe aclarar que dado que este puerto de comunicación fue diseñado para los antiguos modems de línea telefónica, algunos de los nombre y funcionalidad de las señales están definidos para dicho propósito. Para una comunicación bidireccional sin control de handshake se utilizan la salida o transmisión de datos (TxD o Tx), la entrada o recepción de datos (RxD o Rx), y señal de referencia de tensión (GND). Las señales RTS y CTS son de control de flujo o handshake primario, mientras que las DSR y DTR son de handshake secundario.

#pin DB9	**Abrev.**	**Nombre completo**	**Sentido com.**
3	TD	Transmit Data	PC → perif.
2	RD	Receive Data	PC ← perif.
7	RTS	Request To Send	PC → perif.
8	CTS	Clear To Send	PC ← perif.
6	DSR	Data Set Ready	PC ← perif.
5	SG	Signal Ground	PC — perif.
1	DCD	Data Carrier Detect	PC ← perif.
4	DTR	Data Tearminal Ready	PC → perif.
9	RI	Ring Indicator	PC ← perif.

Cuadro 1: Número de pines del conector DB9 y nombres de las señales.

Aún cuando el puerto serie de comunicación no están disponibles en las PC o laptops actuales, es posible disponer del mismo mediante un adaptador USB-serie como los que se muestran en la figura 1. El adaptador de la figura 1a tiene un conector DB9 macho con todas las señales del estándar RS232, mientras que la figura 1b y 1c son conversores USB a niveles digitales de tensión (5V o 3.3V), los cuales tienen en general las señales TxD, RxD, GND y tensión de alimentación.

(a) USB a RS232 (b) USB a TTL con cable (c) USB a TTL sin cable

Figura 1: Adaptadores USB-serie.

[1]IEEE: Institute of Electrical and Electronics Engineers
[2]EIA: Electronic Industries Alliance, RS: Recommended Standard

3. Acceso a dispositivos

En GNU/Linux se tiene acceso a los dispositivos tales como los puertos seriales mediante archivos de dispositivos, los cuales se encuentran en el directorio `/dev`. Los archivos de dispositivos pueden ser de dos tipos: de caracteres y de bloques (indicados con la letra `'c'` y `'b'`). En el listado 1 se muestran tres archivos de dispositivo: dos de dispositivos de caracteres, `ttyUSB0` y `ttyACM0` de conversores USB-serie; y uno de dispositivo de bloques, el disco rígido `sda`. El tipo de dispositivo se ve en la descripción detallada del archivo a la izquierda del listado, tal como `crw-rw----`.

Listing 1: Archivos de dispositivos.

```
crw-rw---- 1 root dialout 188,  0 jul 20 14:02 /dev/ttyUSB0
crw-rw-rw- 1 root dialout 166,  0 jul 20 15:57 /dev/ttyACM0
. . .
brw-rw---- 1 root disk      8,  0 jul 20 14:02 /dev/sda
```

3.1. Dispositivos TTY

El nombre `TTY`, utilizado aún por razones históricas, significa TeleTYpewriter (máquina de escribir remota). En los sistemas de tiempo compartidos anteriores a Unix se utilizaban terminales para interactuar con la computadora. Las terminales generaban los mensajes enviados desde un teletipo, y la información recibida se imprimía localmente. En estos sistemas las terminales se conectaban de forma remota con la computadora mediante el cable adecuado y una UART. En los sistemas operativos (SO) modernos como Unix o GNU/Linux se utilizan las terminales para interactuar con el mismo mediante dispositivos TTY. El funcionamiento interno de las terminales lo maneja directamente el kernel o núcleo del SO. Se puede acceder a las terminales de GNU/Linux presionando las teclas Ctrl+Alt+Fx, con x: 1,. . . ,6.

Un comando útil para ver y modificar la configuración de la terminal es `stty`. El listado 2 muestra la salida del comando `stty` en la terminal actual, y el listado 3 de la terminal asociada al dispositivo `/dev/ttyUSB0`. La opción `-a` es para imprimir toda la información de configuración de la terminal, mientras que la opción `-F` sirve para indicarle el archivo de dispositivo. Se pueden apreciar los valores de algunos parámetros tales como: longitud de palabra, baudrate, control de flujo, paridad, etc.

Listing 2: Configuración de la terminal actual (`/dev/pts/4`).

```
$ stty
speed 38400 baud; line = 0;
-brkint -imaxbel iutf8
```

Listing 3: Configuración del dispositivo serial `/dev/ttyUSB0`.

```
$ stty -a -F /dev/ttyUSB0
speed 9600 baud; rows 0; columns 0; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <
    undef>; eol2 = <undef>; swtch = <undef>; start = ^Q; stop
    = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V;
    discard = ^O; min = 1; time = 0;
-parenb -parodd -cmspar cs8 hupcl -cstopb cread clocal -
    crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr
    icrnl ixon -ixoff -iuclc -ixany -imaxbel -iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0
     tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -
    tostop -echoprt echoctl echoke -flusho -extproc
```

3.2. Comunicación entre dispositivos TTY

Es posible realizar una comunicación entre dos dispositivos TTY directamente desde la terminal de Linux. Para probar esta comunicación se debería disponer de dos PC y dos adaptadores USB-serie conectados entre sí, o bien se puede utilizar la aplicación `socat`. `socat` permite generar dispositivos TTY que se comunican entre sí mediante sockets. Es como disponer de dos puertos seriales virtuales conectados entre sí. Por ejemplo, el comando

```
> socat pty,link=/tmp/ttyS0 pty,link=/tmp/ttyS1
```

crea los archivos de dispositivos TTY `/tmp/ttyS0` y `/tmp/ttyS1`. Se puede acceder a su configuración mediante `> stty -a -F /tmp/ttyS0`. Se puede modificar los parámetros de la comunicación agregando opciones al comando `socat`.

Una vez generados estos archivos, y dejando abierta la terminal donde se encuentra ejecutado `socat`, se puede establecer una comunicación entre las dos terminales creadas. Por ejemplo, al ejecutar

```
> cat /tmp/ttyS1
```

en una terminal, y

```
> echo "Hola TTY" > /tmp/sttyS0
```

en otra, se estará enviando el mensaje `"Hola TTY"` desde `ttyS0` a `ttyS1`.

También existen terminales gráfica como por ejemplo `cutecom` en el SO Linux. En la figura 2 se muestra una ventana de `cutecom` que tiene abierto la terminal `/tmp/ttyS1` con la configuración utilizada por `socat` (velocidad de transmisión de 38400 bps, etc). En la parte principal de la ventana se puede ver el mensaje recibido desde `/tmp/ttyS0` generado con por comando `echo`.

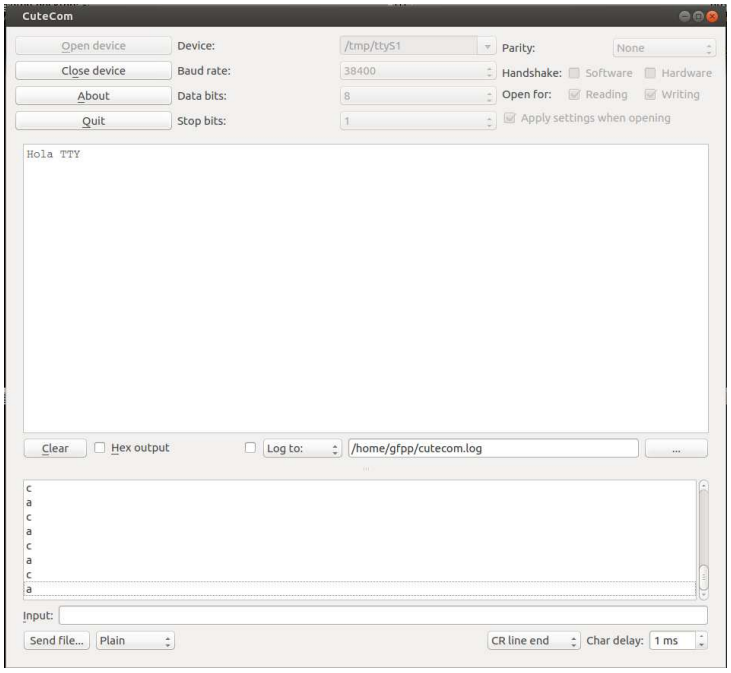

Figura 2: Terminal serial gráfica `cutecom`.

4. Manejo de archivos en lenguaje C

En el SO GNU/Linux casi todo son archivos (excepto los sockets), permitiendo manejar archivos de disco, puerto serie, y otros dispositivos, todos de la misma forma. El procedimiento básico para el manejo de archivo incluye los siguientes pasos: abrir el archivo, leer y/o escribir el archivo, y cerrar el archivo.

4.1. Bajo nivel

Se puede acceder y controlar archivos y dispositivos utilizando algunas funciones de C conocidas como llamadas al sistema. En el núcleo del SO o Kernel están los drivers de dispositivos (device driver) que controlan el hardware a bajo nivel. Los archivos de dispositivos bajo el directorio `/dev` se utilizan como cualquier otro archivo. La funciones C básicas para el manejo de archivo en bajo nivel son: `open()`, `write()`, `read()`, y `close()`. Existe otra función, `ioctrl` que sirve para intercambiar información con el driver del dispositivo.

Al abrir un archivo con `open` o bien crearlo en caso de que no exista (como ser un archivo de texto), la función devuelve un valor entero conocido como descriptor de archivo (file descriptor), el cual sirve para llamar a las demás funciones sobre ese archivo, o sea escribir/leer y cerrarlo.

En Linux todos los programas en ejecución tienen ciertos descriptores de archivos asociados, de los cuales siempre existen los siguientes:

- el descriptor de archivo `0` para la entrada estándar,
- el `1` para la salida estándar, y
- el `2` para la salida de error estándar.

Por lo tanto, se puede imprimir un mensaje a la salida estándar realizando una escritura utilizando el descriptor de archivo asociado, como se muestra en el código fuente del listado 4.

Listing 4: Escritura en el archivo asociado a la salida estándar.

```
1 #include <unistd.h>
2
3 int main(void)
4 {
5   if( write(1, "Hello\n", 6) != 6 )
6     write(2, "Error:␣write()\n", 16);
7
8   return 0;
9 }
```

El programa del listado 4 escribe 6 bytes (caracteres ASCII) a la salida estándar (descriptor de archivo `1`) y verifica que hayan sido escritos correctamente, en caso contrario imprime un mensaje de error en la salida de error. El archivo de cabecera `unistd.h` (estándar de Unix) contiene el prototipo de la función `write()`.

4.2. Alto nivel

Las funciones de manejo de archivos de alto nivel pertenecen a la biblioteca de entrada/salida estándar cuyo archivo de cabecera es `stdio.h`. Se pueden utilizar las funciones de esta biblioteca de la misma forma que las funciones de manejo de archivos de bajo nivel. Al momento de abrir un archivo se devuelve un valor que es utilizado como parámetro para las demás funciones, lo equivalente al descriptor de archivo en bajo nivel en alto nivel se llama *stream* y se implementa como un puntero a una estructura, `FILE*`.

Existen tres streams de archivos abiertos cuando se ejecuta un programa, estos son: `stdin`, `stdout`, y `stderr`. Estos están declarados en `stdio.h` y representan la entrada, salida y salida de error estándares, correspondientes a los descriptores de archivos 0, 1, y 2. Algunas de las funciones de la biblioteca estándar de C para el manejo de archivos en alto nivel son: `fopen()`, `fwrite()`, `fread()`, `fclose()`, `fseek()`, `fflush()`, etc.

En algunas situaciones resultan útiles las funciones de alto nivel en lugar de bajo nivel, como por ejemplo cuando se quiere leer una línea con fin de cadena de la cual no se conoce a priori su longitud.

5. Programación del puerto serie

Como se mencionó anteriormente, dentro del sistema operativo Linux el puerto serie se puede manejar utilizando el archivo de dispositivo de terminal o TTY asociado. La configuración de la terminal, la que

incluye las opciones propias de la comunicación (trama de datos, velocidad de comunicación, control de flujo, etc.) se puede configurar mediante funciones de biblioteca separadas de aquellas utilizadas para la lectura y escritura.

5.1. Programa ejemplo

En el listado 5 se muestra el código fuente para escribir un mensaje en el archivo asociado a un dispositivo serial, en este caso en `/dev/ttyUSB0`. En este programa, primero se abre el archivo con la función `open()`, luego se escribe al archivo con la función `write()`, y finalmente se cierra con la función `close()`. Los argumentos de la función `open()` son: el archivo (con su ruta absoluta), y banderas que indican las propiedades al abrir el archivo. En el ejemplo la bandera `O_WRONLY` abre el archivo para solo escritura.

Listing 5: Escritura en un archivo de dispositivo serial.

```
1  #include <stdio.h>
2  #include <unistd.h> /* Unix standard function definitions */
3  #include <fcntl.h>  /* File control definitions */
4
5  int main(void)
6  {
7    int fd;   /* File descriptor */
8
9    fd = open("/tmp/ttyUSB0", O_WRONLY);
10   if(fd ==  -1)
11   {
12     printf("ERROR: unable to open /tmp/ttyUSB0\n");
13     return -1;
14   }
15
16   write(fd, "Hello TTY\n", 10);
17   close(fd);
18
19   return 0;
20 }
```

El código fuente del listado 5 se puede compilar con[3]

```
> gcc writetty.c -o writetty
```

lo que genera el binario `writetty`. Este se puede ejecutar con

```
> ./writetty
```

Al ejecutar el programa se puede ver el mensaje enviado con `cutecom`, como en la figura 2.

5.2. Configuración de la terminal – estructura termios

`termios` es una interfaz especificada por el estándar POSIX (Portable Operating System Interface). La interfaz de terminal se controla fijando valores a un tipo estructura `termios` y utilizando un pequeño conjunto de funciones, ambos definidos en `termios.h`. Los valores que se pueden manipular para afectar el comportamiento de la terminal están agrupados en varios modos: entrada, salida, control, local, y caracteres especiales. La definición mínima de estructura `termios` se muestra en el listado 6, donde los nombres de los miembros se corresponden con los tipos de parámetros ya mencionados.

Se puede inicializar una estructura `termios` desde la configuración actual llamando a la función `tcgetattr()`, como

```
int tcgetattr(int fd, struct termios * term);
```

[3] `writetty.c` es el nombre de archivo del código fuente que se debe adecuar al caso

Listing 6: Estructura `termios` definida en `termios.h`.

```
struct termios {
  tcflag_t c_iflag;
  tcflag_t c_oflag;
  tcflag_t c_cflag;
  tcflag_t c_lflag;
  cc_t  c_cc[NCCS];
};
```

lo cual carga los valores actuales de configuración de la terminal a la estructura apuntada por el parámetro `term`. A partir de aquí se pueden cambiar los valores de los campos de la estructura y luego configurar la terminal con los valores modificados utilizando la función `tcsetattr()`.

```
int tcsetattr(int fd, int actions, const struct termios * term);
```

El parámetro `actions` controla cómo se aplican los cambios, pudiendo ser:

- `TCSANOW`: cambiar los valores inmediatamente.
- `TCSADRAIN`: cambiar los valores cuando la salida actual este completa.
- `TCSAFLUSH`: cambiar los valores cuando la salida actual este completa, pero descartando cualquier entrada disponible y que no ha sido leída aún por una llamada a `read()`.

5.3. Módulo para configuración del puerto serie

Para poder utilizar una función de configuración de puerto serie en diferentes aplicaciones es conveniente tener un módulo separado de dicha implementación. Este módulo está comprendido por un archivo de cabecera del listado 7 con la declaración de la función de configuración, y un archivo de código fuente del listado 8 con la correspondiente implementación. En esta implementación, la función `termset` configura la comunicación serie con la trama más común de comunicación 8N1 (8 bits de datos, sin bit de paridad y 1 bit de stop), habilitado para lectura y sin control de flujo por hardware ni por software (XON/XOFF).

Listing 7: Archivo de cabecera del módulo, `termset.h`.

```
#ifndef _TERMSET_H
#define _TERMSET_H

#include <termios.h>

struct termios ttyold, ttynew;

/* termset function
 * Parameters:
 *  fd: file descriptor -device- (ex: /dev/ttyUSB0)
 *  baudrate: communication speed (ex: 9600, 115200)
 *  ttyold: current termios structure
 *  ttynew: new termios structure
 */
int termset(int fd, int baudrate, struct termios * ttyold,
  struct termios * ttynew);

#endif
```

6. Ejemplos con Arduino

A continuación se muestran algunos ejemplos de código fuente de programación de puerto serie para comunicar la PC con un Arduino, como también algunos ejemplos de sketchs Arduinos para tal fin. Al

Listing 8: Archivo de código fuente del módulo, `termset.c`.

```c
#include <stdio.h>
#include "termset.h"

int termset(int fd, int baudrate, struct termios * ttyold, struct termios * ttynew)
{
  switch(baudrate)
  {
    case 115200: baudrate = B115200;
                 break;
    case 57600:  baudrate = B57600;
                 break;
    case 38400:  baudrate = B38400;
                 break;
    case 19200:  baudrate = B19200;
                 break;
    case 9600:   baudrate = B9600;
                 break;
    default:     baudrate = B115200;
                 break;
  }

  if(tcgetattr(fd, ttyold) != 0)
  {
    printf("ERROR: tcgetattr\n");
    return -1;
  }
  ttynew = ttyold;

  cfsetospeed(ttynew, baudrate);
  cfsetispeed(ttynew, baudrate);

  ttynew->c_cflag = (ttynew->c_cflag & ~CSIZE) | CS8;// 8 data bits (8)
  ttynew->c_cflag &= ~(PARENB | PARODD);             // no parity   (N)
  ttynew->c_cflag &= ~CSTOPB;                        // 1 stop bit  (1)

  ttynew->c_cflag |= (CLOCAL | CREAD);// ignore modem status lines, and
                                      // enable reading
  ttynew->c_cflag &= ~CRTSCTS;        // no flow control

  ttynew->c_iflag &= ~IGNBRK;         // disable break processing
  ttynew->c_iflag &= ~(IXON | IXOFF | IXANY); // shut off xon/xoff ctrl

  ttynew->c_lflag = 0;                // no signaling chars, no echo,
  ttynew->c_oflag = 0;                // no remapping, no delays
  ttynew->c_cc[VMIN]  = 0;            // read doesn't block
  ttynew->c_cc[VTIME] = 100;          // read timeout

  /*
   * TCSANOW: Make the change immediately.
   * TCSADRAIN: Make the change after all queued output has been written.
   * TCSAFLUSH: This is like TCSADRAIN, but also discards any queued input.
   */
  if(tcsetattr(fd, TCSAFLUSH, ttynew) != 0)
  {
    printf("ERROR: tcsetattr\n");
    return -1;
  }

  return 0;
}
```

conectar un Arduino a la PC con sistema operativo Linux se genera un dispositivo serial en `/dev/ttyACM0` utilizado por el IDE Arduino para grabar el programa, y pudiendo también ser utilizado para comunicar la PC con el Arduino.

El listado 9 muestra el código fuente de un sketch Arduino que permite encender y apagar LEDs

mediante el envío de letras por el puerto serie[4]. El programa se puede probar utilizando el *Monitor Serie* del IDE Arduino o bien se puede programar una aplicación que permita enviar las letras correspondientes. Al enviar la letra `'v'` se conmuta el estado del LED verde conectado a `PIN 5`, la letra `'a'` para el LED amarillo conectado al `PIN 6`, y la letra `'r'` para el LED rojo conectado al `PIN 9`. El listado 10 muestra el código fuente donde se escribe el caracter `'v'` por el puerto serie, que al ejecutar desde la PC conectada al Arduino y con el sketch anterior grabado, hace que el LED verde se encienda y apague cada un segundo aproximadamente. Se puede compilar el programa con

```
> gcc writechar.c termset.c -o writechar
```

el cual utiliza también el código fuente del módulo de configuración de puerto serie.

El listado 11 muestra un código fuente de ejemplo de una aplicación para leer el puerto serie mediante funciones de alto nivel de manejo de archivos. Este código se puede utilizar para leer una cadena de texto enviada desde un Arduino al cual se le ha grabado el sketch de ejemplo que envía el valor del conversor analógico a digital (ADC, Analog to Digital Converter). Este ejemplo se encuentra en el IDE Arduino en `File->Examples->01.Basics->AnalogReadSerial`. Dado que el valor del ADC se envía como número entero en formato de cadena de longitud variable, es conveniente aquí utilizar las funciones de manejo de archivo de alto nivel como se indicó anteriormente, particularmente la función `getline()`. Se puede compilar el programa con

```
> gcc readline.c termset.c -o readline
```

generando el binario `readline`, el cual se debe ejecutar con

```
> ./readline /dev/ttyACM0 9600
```

El programa recibe desde la terminal el dispositivo serie a utilizar y la velocidad de comunicación.

El listado 12 muestra un sketch Arduino que toma el valor del ADC y lo envía por puerto serie en diferentes formatos. El Arduino, al recibir el caracter `'c'` envía el valor como cadena de caracteres, al recibir `'e'` lo envía como número entero (4 bytes), y al recibir `'f'` lo envía como número en punto flotante (4 bytes de la representación `float`). Se puede utilizar el monitor serial del IDE Arduino para probar la recepción cuando se solicita el valor como cadena. Sin embargo, para observar los valores en representación entero o punto flotante es conveniente utilizar `cutecom` y ver los valores en representación hexadecimal, o bien desarrollar un programa C para realizar dicha lectura y conversión de tipo. El listado 13 muestra el código fuente para realizar la lectura por puerto serie de un valor entero (`int`) recibido en formato binario por puerto serie.

El listado 14 muestra un sketch Arduino que envía por el puerto serie el estado de tres pulsadores que están conectados a los pines 4, 7 y 8.

6.1. Ejercicios de programación de puerto serie en C

A continuación se enuncian algunos ejercicios para elaborar programas en lenguaje C de programación de puerto serie que interactúen con un Arduino utilizando los sketchs antes vistos.

1. Programa de comunicación con Arduino con el sketch del listado 9 que:
   - Abra el puerto serie `/dev/ttyACM0` a 9600 bps.
   - Solicite al usuario de forma continuada presionar: `'v'` para cambiar el estado del LED verde, `'a'` para el LED amarillo, `'r'` para el LED rojo, hasta ingresar `'s'` para salir.
   - Cierre el puerto serie.
2. Programa de comunicación con Arduino con el sketch del listado 14 que:
   - Abra el puerto serie `/dev/ttyACM0` a 9600 bps.
   - Lea de forma continuada una cadena de 4 caracteres (3 estados de pulsadores y retorno de carro) y la imprime en la salida estándar.

[4]En https://github.com/ciiiutnfrc/ponchitoCIII se encuentra disponible un ponchito o shield para Arduino adecuado para esta aplicación.

Listing 9: Sketch Arduino para manejo de LEDs por puerto serie.

```
/*
 * Recibe una letra por puerto serie para cambiar el estado de los LEDS.
 * 'v': cambia el estado del LED verde.
 * 'a': cambia el estado del LED amarillo.
 * 'r': cambia el estado del LED rojo.
 *
 * Se puede probar con el "Monitor serie" del IDE Arduino.
 *
 * El código de este ejemplo es de dominio público.
 */

#define GPIO_LED_VERDE    5
#define GPIO_LED_AMARILLO 6
#define GPIO_LED_ROJO     9

// La función 'setup' se ejecuta una única vez al presionar reset o al
// encender la placa.
void setup() {
  // Inicializa el puerto serie (UART) a 9600 bps.
  Serial.begin(9600);

  // inicialización de los pines GPIO como salida para los LEDs.
  pinMode(GPIO_LED_VERDE, OUTPUT);
  pinMode(GPIO_LED_AMARILLO, OUTPUT);
  pinMode(GPIO_LED_ROJO, OUTPUT);
}

// La función 'loop' corre indefinidamente una y otra vez.
void loop() {
  size_t n;
  uint8_t letra[1];

  // Lee la letra (1 byte) por puerto serie.
  n = Serial.readBytes(letra, 1);
  if(n == 1)
  {
    switch(letra[0])
    {
      case 'v':
        toggle(GPIO_LED_VERDE);
        break;
      case 'a':
        toggle(GPIO_LED_AMARILLO);
        break;
      case 'r':
        toggle(GPIO_LED_ROJO);
        break;
    }
  }
}

// Función para cambiar el estado de un LED (GPIO).
void toggle(uint8_t gpio)
{
  if(digitalRead(gpio) == HIGH)
    digitalWrite(gpio, LOW);
  else
    digitalWrite(gpio, HIGH);
}
```

3. Programa de comunicación con Arduino con el sketch del listado 12 que:

   - Abra el puerto serie pasado desde la línea de comandos a 9600 bps.
   - Solicite al usuario de forma continuada presionar: `'c'` para solicitar el valor del ADC como cadena, `'e'` para solicitar el valor como entero, `'f'` como punto flotante, e imprimir el valor

Listing 10: Escritura de un caracter por puerto serie.

```c
#include <stdio.h>     /* Standard input/output definitions */
#include <fcntl.h>     /* File control definitions */
#include <unistd.h>    /* UNIX standard function definitions */
#include "termset.h"

int main(void)
{
  int fd; /* Descriptor de archivo del puerto */
  struct termios oldtty, newtty;

  fd = open("/dev/ttyACM0", O_RDWR | O_NOCTTY | O_NDELAY);
  if(fd == -1)
  {
    printf("ERROR: no se pudo abrir el dispositivo.\n");
    return -1;
  }
  if(termset(fd, 9600, &oldtty, &newtty) == -1)
  {
    printf("ERROR: no se pudo configurar el TTY\n");
    return -1;
  }

  tcflush(fd, TCIOFLUSH);
  for(;;)
  {
    write(fd, "v", 1);
    tcdrain(fd);
    sleep(1);
  }

  close(fd);
  return 0;
}
```

recibido.

- Termine la solicitud cuando se ingrese `'s'`.
- Cierre el puerto serie.

Listing 11: Lectura de archivo de alto nivel.

```c
#include <stdio.h>
#include <stdlib.h>
#include "termset.h"

int main(int argc, char * argv[])
{
  int fd;
  FILE * fptr;
  ssize_t n_read;
  size_t len = 0;
  char * line;

  if(argc != 3)
  {
    printf("USAGE: %s <device> <baudrate>\n", argv[0]);
    return -1;
  }

  fptr = fopen(argv[1], "r");
  if(fptr == NULL)
  {
    printf("ERROR: opening %s file\n", argv[1]);
    return -1;
  }
  fd = fileno(fptr);

  if(termset(fd, atoi(argv[2]), &ttyold, &ttynew) == -1)
  {
    printf("ERROR: setting tty\n");
    return -1;
  }

  for(;;)
  {
    n_read = getline(&line, &len, fptr);
    if(n_read != -1)
      printf("Line: %s", line);
  }

  fclose(fptr);
  return 0;
}
```

Listing 12: Sketch Arduino que envía el valor del ADC por puerto serie.

```
/*
 * Envía por puerto serie el valor del ADC en diferentes formatos por petición.
 * 'c': envía el valor entero como cadena ("0000" a "1023").
 * 'e': envía el valor como nro. entero.
 * 'f': envía el valor del voltaje como nro. de punto flotante (float).
 *
 * Se puede probar con el "Monitor serie" del IDE Arduino.
 *
 * El código de este ejemplo es de dominio público.
 */

#define CANAL_ADC A0

// Union de entero y unsigned char.
typedef union
{
  uint8_t uc[4];
  int i;
} int_uc_t;

// Union de float y unsigned char.
typedef union
{
  uint8_t uc[4];
  float f;
} float_uc_t;

// Variables globales.
size_t n;
uint8_t solicitud;

char cadena[5];   // Valor del ADC como cadena.
int_uc_t i_uc;    // Valor del ADC en entero.
float_uc_t f_uc;  // Valor del ADC en punto flotante (float).

// La función 'setup' se ejecuta una única vez al presionar reset o encender la placa.
void setup() {
  // Inicializa el puerto serie (UART) a 9600 bps.
  Serial.begin(9600);
}

// La función 'loop' corre indefinidamente una y otra vez.
void loop() {
  // Lectura de la entrada analógica.
  int valor_adc = analogRead(CANAL_ADC);

  // Lee la letra (1 byte) por puerto serie.
  n = Serial.readBytes(&solicitud, 1);

  if(n == 1)
  {
    switch(solicitud)
    {
      // Envia el valor del ADC como cadena.
      case 'c':
        entero_a_cadena(valor_adc, 4, cadena);
        Serial.write(cadena, 5);
        break;
      case 'e':
        i_uc.i = (int)valor_adc;
        Serial.write(i_uc.uc, 4);
        break;
```

Listing 12 (cont.): Sketch Arduino que envía el valor del ADC por puerto serie.

```
63        case 'f':
64          f_uc.f = valor_adc * (5.0 / 1023.0);
65          Serial.write(f_uc.uc, 4);
66          break;
67      }
68    }
69    delay(100);
70  }
71
72  void entero_a_cadena(unsigned int entero, unsigned char cant_digitos,...
73    char * cadena)
74  {
75    unsigned int n, res;
76
77    for(n = 0; n < cant_digitos; n++)
78    {
79      res = (entero / potencia_entero(10, cant_digitos-n-1));
80      cadena[n] = res + '0';
81      entero -= res * potencia_entero(10, cant_digitos-n-1);
82    }
83    cadena[cant_digitos] = '\0';
84  }
85
86  int potencia_entero(int x, int n)
87  {
88    if(n == 0)
89      return 1;
90
91    int r = 1;
92    while(n--)
93      r *= x;
94
95    return r;
96  }
```

Listing 13: Lectura de valor entero por puerto serie.

```c
#include <stdio.h>    /* Standard input/output definitions */
#include <fcntl.h>    /* File control definitions */
#include <unistd.h>   /* UNIX standard function definitions */
#include "termset.h"

typedef union
{
  unsigned char uc[4];
  int i;
} int_uchar_t;

int main(void)
{
  int fd; /* Descriptor de archivo del puerto */
  size_t n = 0;
  unsigned char byte = 0;
  struct termios oldtty, newtty;

  int_uchar_t i_uc;

  fd = open("/dev/ttyACM0", O_RDWR | O_NOCTTY | O_NDELAY);
  if(fd == -1)
  {
    printf("ERROR: no se pudo abrir el dispositivo.\n");
    return -1;
  }
  termset(fd, 9600, &oldtty, &newtty);

  tcflush(fd, TCIOFLUSH);
  for(;;)
  {
    write(fd, "e", 1);
    tcdrain(fd);
    usleep(250000);

    n = read(fd, &(i_uc.uc[0]), 4);
    printf("Valor entero: %d\n", i_uc.i);
  }

  close(fd);
  return 0;
}
```

Listing 14: Envía por puerto serie el estado de las teclas.

```
/*
 * Envía los estados de las entradas (pulsadores) por puerto serie (UART).
 * Se puede probar con el "Monitor serie" del IDE Arduino.
 *
 * El código de este ejemplo es de dominio público.
 */

#define GPIO_BOTON_IZQ 8
#define GPIO_BOTON_MED 7
#define GPIO_BOTON_DER 4

// La función 'setup' se ejecuta una única vez al presionar reset o encender la placa.
void setup() {
  // Inicializa el puerto serie (UART) a 9600 bps.
  Serial.begin(9600);

  // Configura los GPIO de los pulsadores como entradas.
  pinMode(GPIO_BOTON_IZQ, INPUT);
  pinMode(GPIO_BOTON_MED, INPUT);
  pinMode(GPIO_BOTON_DER, INPUT);
}

// La función 'loop' corre indefinidamente una y otra vez.
void loop() {
  // Lee el valor de los pines de entrada (GPIO).
  int boton_izq = digitalRead(GPIO_BOTON_IZQ);
  int boton_med = digitalRead(GPIO_BOTON_MED);
  int boton_der = digitalRead(GPIO_BOTON_DER);

  // Envía por puerto serie el estado del pulsador.
  Serial.write(boton_izq + '0');
  Serial.write(boton_med + '0');
  Serial.write(boton_der + '0');
  Serial.write('\n');

  delay(100);
}
```