Sistemas Informáticos Industriales

Apuntes de

Programación para las comunicación serie.

Grado en Electrónica y Automática
Departamento de Informática de Sistemas y Computadores
Escuela Técnica Superior de Ingeniería del Diseño

Ángel Perles
programacion_comunicaciones_serie_r02.odt

Contenido

Índice de contenido

8 PROGRAMACIÓN PARA LAS COMUNICACIONES SERIE

7.1 INTRODUCCIÓN

Las normas de comunicación serie, como la RS-232 y la RS-485, permiten aprovechar muchos dispositivos industriales desde aplicaciones para computador.

En este tema se presentan dichas normas, que corresponden al nivel físico (1) de referencia del estándar OSI. Sobre dichas normas se pueden construir nuevas capas que aporten otras funcionalidades.

Este tema está pensado para trabajarlo de forma lineal sin necesidad de acudir a otras fuentes ni materiales adicionales.

En puntos concretos se intercalan actividades que permitirán practicar los conocimientos adquiridos.

En caso de dificultad en la resolución de actividades sí se recomienda acceder a fuentes externas, por ejemplo, a la bibliografía recomendada.

7.2 OBJETIVOS

- Aprender a desarrollar aplicaciones que hagan uso de conexiones serie.
- Aplicar técnicas de comunicación serie a dispositivos industriales.

7.3 PROGRAMACIÓN DE LA INTERFAZ EN EL PC

El PC solía disponer de 1 ó 2 conexiones serie tipo RS-232. El conector exterior suele ser del tipo DB-9 macho, pero también puede ser un conector DB-25 macho.

Este tipo de interfaz tiende a desaparecer en los PC de sobremesa y portátiles para el hogar (está disponible en placa base, pero no tiene puesto el conector), y se mantiene en los PC de tipo negocio e industriales. En caso de no disponer de ella, se puede utilizar un conversor USB-serie disponible en el mercado.

7.3.1USO COMO ARCHIVOS

Los sistemas operativos suelen ofrecer servicios que permiten usar más fácilmente este tipo de puertos.

Una de las formas estándar de tratar los puertos serie es como unos ficheros en los que las órdenes de escritura envían caracteres y las de lectura permiten recoger los caracteres recibidos. Tanto MS-DOS/WINDOWS como Unix proporcionan este tipo de servicio.

Desde el punto de vista de los sistemas operativos MS-DOS y WINDOWS, los puertos serie son dispositivos de tipo fichero que se nombran utilizando las palabras clave COM1, COM2, COM3… COMn.

En el caso de sistemas tipo *nix (Linux, MacOSX, Android, Solaris, ...), los dispositivos de este tipo se suelen nombrar como /dev/ttyS0, /dev/ttyS1 ...

Como ejemplo, suponiendo una conexión serie correctamente configurada, se podría trasmitir el texto “Hola Mundo” de la siguiente manera en C:

```
#include <stdio.h>
#include <stdlib.h>

int main (void) {
    FILE *port;

    port = fopen("COM1:","w");          // Windows
    // port = fopen("/dev/ttyS0","w"); // *nix
    if (port == NULL) {
        printf("Algo falla\n");
        exit(1);
    }

    fprintf(port,"Hola, Mundo\n");
    fclose(port);

    return(0);
}
```

7.3.2PROPUESTA DE USO PARA QT

Recogido en el tutorial “ El puerto serie y Qt”

http://www.disca.upv.es/aperles/qt/port_serie_Qt/port_serie_Qt.html

Una versión algo más trabajada

http://riunet.upv.es/handle/10251/30515

NOTA: Qextserial port ha sido ustituido por qtserialport. Estoy preparando un borrador nuevo de tutorial.

7.4 GESTIÓN GENERAL DE CONEXIONES SERIE

7.4.1 CONCEPTOS

La norma RS-232 es utilizada en una gran variedad de dispositivos de dos maneras no excluyentes:

- **Configuración del dispositivo**. Para establecer el comportamiento del dispositivo.
- **Trabajo normal**. El dispositivo trabajará en colaboración con otro dispositivo, recibiendo y/o enviando información durante el trabajo normal.

Un ejemplo típico del primer caso sería la programación de un autómata programable, donde, en una primera fase, se transfiere a través de una conexión serie el programa de control que ejecutará el autómata.

Un ejemplo del segundo caso sería un módulo GPS conectado a través de su salida serie a un sistema de seguimiento de vehículos.

El uso habitual de los terminales serie y de emuladores de terminal hace que, muchas veces, la información transferida entre los dos equipos sea textual y con significado fácilmente comprensible por el usuario. De lo dicho se deduce que hay dos modalidades de transferir datos en RS-232: ***información binaria*** o ***información en forma de cadenas de texto*** (en realidad la segunda es un subconjunto de la primera).

Desde el punto de vista del programador, ambas aproximaciones son equivalentes, la diferencia es que, en el segundo caso, la interpretación humana del significado es más sencilla.

El aprovechamiento de la conexión serie consistirá entonces en saber generar las secuencias a enviar por el canal serie y en interpretar dichas secuencias, aspecto que se trata en este apartado.

7.4.2 ENVÍO DE INFORMACIÓN

Partiendo de que la conexión serie del computador está abierta y configurada adecuadamente, supóngase que se tiene una función general que permiten usar la conexión serie y cuyo prototipo es:

```
void serie_sendData(const unsigned char *datos,
                    int cant_datos);
```

La función `serie_sendData()` deposita en el buffer de transmisión del sistema operativo (es decir, no espera a que termine la transmisión de datos) una secuencia de `cant_datos` datos que serán transmitidos uno a uno a través del canal serie.

La siguiente figura representa la idea.

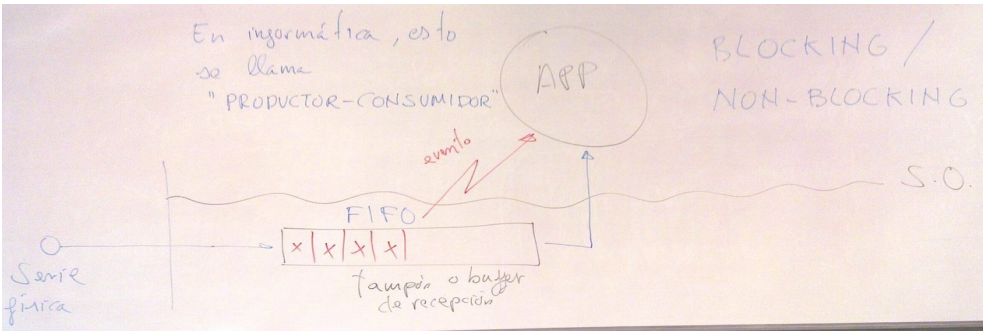

Ilustración 1: Gestión productor-consumidor de la interfaz serie por parte del SO

Destacar que, en función de la configuración del número de bits, los bits de mayor peso de los bytes pueden ser descartados.

Por ejemplo, para transmitir los datos 125, 33, 40, 100 en ese orden se podría hacer:

```
unsigned char info[] = {125, 33, 40, 100};

serie_sendData( info, 4);
```

Para transmitir la cadena “Hola” se podría hacer:

```
char *cad = ”Hola”;
serie_sendData(cad, strlen(cad));
```

Actividad.

Se desea controlar una red industrial basada en RS-485 (suponemos ahora que es lo mismo que RS-232) usando un PC. Cada módulo conectado a la red tiene un identificador diferente, y se desea acceder a un módulo de salidas digitales que acepta el formato de mensaje abajo mostrado.

Diseñar una función que haga uso de la función `sendData()` y construya un mensaje que envíe por el serie a partir de los parámetros que se le pasan como argumento.

Formato mensaje:

#mmssvccc
- **mm** → identificador de módulo, número decimal de 2 cifras
- **ss** → salida a modificar, número decimal de 2 cifras
- **v** → valor de la salida (“0” o “1”)
- **ccc** → suma de comprobación, la suma de mm, ss y v con 3 cifras decimales

Ejemplo: #07031011 → módulo 7, salida 3, valor 1, suma de comprobación 11

La función a rellenar es

```
void maneja_modulo(int num_modulo, int num_salida, int va-
lor) {

}
```

Y un ejemplo de uso:

```
maneja_modulo(7,3,1);
```

7.4.3RECEPCIÓN Y TRATAMIENTO DE INFORMACIÓN

Partiendo de que el puerto serie del computador está abierto y configurado adecuadamente, supóngase que se tiene una función general que permite usar la conexión serie para recibir datos y cuyo prototipo es:

```
int serie_receiveData(unsigned char *datos, int max_datos);
```

Esta función copia en la dirección indicada por el puntero los datos recibidos a través del serie y devuelve la cantidad de datos recibidos. Si no hubiese datos disponibles, esta función devolvería un valor 0 (es decir, en jerga informática, esta función es “no bloqueante”).

Para prevenir un rebasamiento del buffer destino, el parámetro `max_datos` permite limitar la cantidad máxima de datos a copiar, quedando el resto en el buffer del SO.

Para recoger la información recibida del serie, independientemente de si es texto o datos, se podría hacer,

```
unsigned char buffer[1000];  //deposito datos
int num_datos;

num_datos = serie_receiveData( buffer, 999);
```

Si los datos son textuales, podríamos mostrarlos por pantalla de la siguiente forma,

```
buffer[n_datos] = '\0'; //para que sea cadena C
printf("Se ha recibido %s\n", buffer);
```

Estas funciones son lo suficientemente genéricas como para que puedan aplicarse a cualquier software que utilice un flujo de datos. Más aún, serían aplicables a cualquier sistema que se basase en flujos de datos (USB, conexiones TCP/IP, etc.).

Las técnicas de procesamiento de canales consisten en:

1. Acumular en un buffer de entrada las secuencias de datos recibidas.
2. Buscar la completitud de “mensajes”.

- SI encontrado: Interpretar y eliminar mensaje.
- NO encontrado: Continuar con 1.

En este caso, un mensaje será la secuencia de caracteres completa que transporta un item de información.

El formato de un mensaje dependerá del dispositivo particular. Pero será claramente distinguible y separable del resto de mensajes dentro del flujo. Formas habituales de mensajes son:

- Empezar y acabar con caracteres especiales. Por ejemplo, ...”$628689652896#”.
- Acabar el mensaje con un retorno de carro. Por ejemplo, “yuwtw5736fkj\x0d” o “gfd7g9s79\x0D\x0A”.
- Usar un tamaño fijo de mensaje. Por ejemplo, “...abcdabcdabcd...”.

El tratamiento del flujo consistirá en ir localizando y procesando los mensajes a medida que estos llegan. En el tratamiento se deberá tener especial precaución en:

- La localización de un mensaje no debe depender de la llegada de futuros mensajes.
- El arranque del flujo de datos puede iniciarse en cualquier punto. No suponer nunca que siempre van a llegar mensajes completos.
- En una aplicación profesional, la consistencia del mensaje debe comprobarse siempre (longitud, formato, checksums, etc.).
- Hay que considerar la introducción de “timeouts” y de mecanismos de recuperación de inconsistencias.

Actividad

Un módulo industrial de entradas digitales emite continuamente la secuencia de caracteres mostrada abajo, donde el carácter numérico de más a la derecha corresponde a la entrada 1 y el de más a la izquierda a la entrada 8.

Los caracteres "H" y "L" representan un nivel alto y bajo respectivamente en la entrada correspondiente.

Suponiendo que se conecta el módulo a una entrada serie de un PC y se usa la función `receiveData()`, desarrollar el programa que analiza los caracteres recibidos y, en función del valor de la entrada 7, poner la propiedad Visible a true (para un 1) o a false (para un 0) de un objeto llamado “Puerta”.

...*HLLHHLHL...

Actividad.

Adapta las actividades anteriores al componente TComPort.

En el segundo caso se deberá emplear el evento de recepción de caracteres.

7.4.4EJEMPLO TRATAMIENTO GPS NMEA-0183

La mayoría de los GPS comerciales utilizan la especificación estándar NMEA-0183 (National Marine Electronics Association) para generar información con destino al dispositivo consumidor (un ordenador, un móvil, etc.).

(En la pizarra y en el proyector hemos visto el formato típico de mensaje de un GPS, en los ejemplos de la web tenéis los enlaces para verlo, no lo pongo aquí este año).

Como ejemplo general de tratamiento del flujo se utilizará este tipo de dispositivo y se dan las pautas a seguir.

Supóngase una conexión serie abierta y correctamente configurada.

La primera etapa a resolver consiste en proporcionar un mecanismo que permita ir recibiendo el flujo de datos y acumulándolo para su posterior análisis.

Como ejemplo de esta etapa, se propone crear un vector de datos suficientemente granada e ir leyendo la información del canal serie y acumulándola.

Cada vez que se reciban nuevos datos, esto se acumulan y se analizan para localizar los mensajes. Cuando un mensaje es localizado, se puede proceder a eliminarlo del buffer.

El siguiente fragmento de código es una posible implementación de esta idea.

```
#define TAMANYO_BUFFER 20000
static char buffer[TAMANYO_BUFFER]; // buffer de recepción
static int total_datos;             // contador del número de datos recibidos

//-- limpiar el buffer de recpción de datos del serie ----------------------
void gps_inicializar(void) {
    total_datos = 0;
}

//-- ir recibiendo la información del puerto serie, acumularla y llamar al tratamiento
void gps_procesar(void) {
    int num_datos;

    // recibir los datos
    num_datos = serie_receiveData(buffer+total_datos,TAMANYO_BUFFER-(total_datos-1));

    if (num_datos > 0) {
         // actualizar cuenta de datos
        total_datos = total_datos + num_datos;

        // buscar y tratar mensajes
        bool mensaje_encontrado;
        do {
            int final_mensaje;

            mensaje_encontrado = gps_mensaje_buscar(buffer,total_datos,&final_mensaje);

            if (mensaje_encontrado) {   // eliminar del buffer el mensaje, con
independencia de que sea correcto o no
                // eliminar el mensaje de buffer, se podría hacer con strcpy si
pusiesemos \0 al final
                memmove(buffer,buffer+final_mensaje+1,total_datos-(final_mensaje+1));
                total_datos = total_datos-(final_mensaje+1);
            }
        } while (mensaje_encontrado);
    }
}
```

La función gps_procesar() debe ser llamada cada cierto tiempo para que vaya recogiendo los datos acumulados en el buffer interno del sistema operativo.

La tarea de localizar los mensajes la dejamos en manos de otra función, gps_mensaje_buscar(). Su cometido será intentar localizar las marcas apropiadas dentro del buffer que delimitan un mensaje. El mensaje localizado puede ser correcto o corrupto, pero, sea del tipo que sea, deberá ir purgándose nuestro buffer de recepción.

En el caso del GPS NMEA, la mejor opción para localizar el mensaje es, quizá, localizar el carácter de finalización del mensaje, que es el número 0Ah (en C, el carácter '\x0A'). Encontrado éste, se sabe que hay un mensaje o un fragmento corrupto de uno.

Para tener la certeza de que tenemos localizado el mensaje completo, se puede retroceder en la búsqueda y localizar el carácter de inicio del mensaje, que es el símbolo '$'.

El siguiente fragmento de código es una posible implementación de la idea.

```
//-- analizar el buffer de recepción para localizar un posible mensaje --------
// y tratarlo en el caso de que esté completo
bool gps_mensaje_buscar(char *buff, int tot_datos,int *fin_mensaje) {

    bool encontrado;

    encontrado = false;

    for (int i=0;i<tot_datos; i++) {
        if (buff[i]=='\x0A') {  // localizar final del mensaje
            encontrado = true;  // se ha encontrado el final de un mensaje
            *fin_mensaje = i;
            break;  // salir del for
        }
    }

    // si se ha encontrado el final, tratar de encontrar el principio
    if (encontrado) {
        for (int i=(*fin_mensaje);i>0;i--) {
            if (buff[i]=='$') { // localizar el principio del mensaje
                gps_mensaje_interpretar(buff+i,(*fin_mensaje)-i+1);   // mensaje
completo, tratarlo
                break;
            }
        }
    }

    // salimos
    return (encontrado);
}
```

Esta función devuelve “verdadero” si se ha encontrado el final del mensaje y deposita en un puntero el lugar donde se a encontrado ese final. Esto permite al nivel superior la eliminación del mensaje del buffer de entrada.

Cuando se localiza el mensaje completo, se pasa a la etapa de análisis del mensaje.

Como ejemplo, la función gps_mensaje_interpretar() se encargará de analizar primero la consistencia del mensaje, y, si es correcto, extraer la información de interés. En este caso se desean extraer las coordenadas de posición del GPS.

El siguiente fragmento de código es una posible implementación:

```
// interpretar el contenido del mensaje  -------------------------------------
void gps_mensaje_interpretar(char *buffer, int tamanyo) {

    unsigned char check2;
    unsigned int check1;
    double longitud, latitud;
    char msg[100];

    // Comprabamos consistencia del mensaje.
    // Para NMEA-0183 el mensaje termina con un checksum
    if (tamanyo < 6) { //huy, mensaje muy corto, no puede ser
        return;
    }

    // vemos que está el checksum
    if ((buffer[tamanyo-5]!='*')||(!isxdigit(buffer[tamanyo-4]))||(!
isxdigit(buffer[tamanyo-3]))) {
        qDebug("Esto no tiene checksum");
        return;
    }

    // ahora calculamos y comparamos los checksum
    sscanf((char *)(buffer+tamanyo-4),"%2x",&check1);
    check2 = 0;
    for (int i=1;i<tamanyo-5;i++) {
        check2 = check2 ^ buffer[i];
    }

    if (check1 != check2) { // vaya, mensaje corrupto
        qDebug("Mensaje hecho polvo");
        return;
    }

    // interpretar mensaje, por ejemplo ...
    // $GPGGA (Global Positioning System Fix Data)
    // i.e. $GPGGA,170834,4124.8963,N,08151.6838,W,1,05,1.5,280.2,M,-34.0,M,,,*75
    // ver http://home.mira.net/~gnb/gps/nmea.html#gpgga
    if (strncmp((const char*)buffer+1,"GPGGA",5)==0) { // es el que buscamos
        const char *p;

        p = (const char *)(buffer+7);
        p=strchr(p,',');p++;    // buscar la siguiente coma y pasarla
        sscanf(p,"%lf",&latitud);
        latitud = latitud / 100.0;   // pasar a grados
        p=strchr(p,',');p++;    // buscar la siguiente coma y pasarla
        p=strchr(p,',');p++;    // buscar la siguiente coma y pasarla
        sscanf(p,"%lf",&longitud);
        longitud = longitud /100.0;  // pasar a grados

        // faltaria sacar norte-sur, este-oeste, otro día

        printf("Estamos en %lf grados de latitud y %lf grados de
longitud",latitud,longitud);
    }
}
```

7.5 BIBLIOGRAFÍA Y ENLACES