

Práctica 1: El puerto serie

Emilio Cobos Álvarez (B1)
Anna Elena Chesnais (B1)
Cristina García González (B1)

6 de octubre de 2015

Índice

A. El cable de módem nulo	3
1. Construya un cable de módem nulo con el fin de comunicar dos equipos por el puerto serie (recomendada versión completa).	3
a. Use el programa <i>PuTTY</i> para comprobar dicha comunicación. ¿Qué parámetros se pueden modificar en el puerto serie?. ¿Qué valores son posibles en cada uno de ellos?. Pruebe distintas configuraciones (al menos dos) y describa lo que ocurre al teclear caracteres en una y otra consola. ¿Se puede seleccionar el conjunto de caracteres que el terminal remoto está utilizando?. ¿Para qué sirve esto?.	3
2. Modifique la configuración del puerto serie entre 8 y 7 bits de datos	4
a. Justifique lo que ocurre al transmitir en ambos casos la letra ñ	4
3. Pruebe los programas de ejemplo síncrono y asíncrono para Windows y Linux	4
a. Comentar el funcionamiento de los ejemplos facilitados	4
b. Especifique los parámetros de configuración del puerto serie en cada uno de los ejemplos e indique también la/s línea/s de código en la que se configuran	4
c. Comentar brevemente el API de Windows y Linux para configurar el puerto serie, y cómo se lee y escribe.	5
d. ¿En qué se diferencia el modelo síncrono del asíncrono?	5
e. ¿Qué ocurre si cruzamos el programa de ejemplo asíncrono para Windows con el de Linux? Justifique la respuesta.	5
4. Cree un fichero de texto con nombre serie.ini como el que figura a continuación:	5
a. Modifique y adjunte el programa de ejemplo asíncrono para Windows de forma que cargue los datos de configuración del puerto serie de este fichero. Utilice las funciones de Windows <code>GetPrivateProfileString</code> y <code>GetPrivateProfileInt</code> para recuperar directamente los valores del archivo de configuración.	5
b. Verifique su correcto funcionamiento mostrando la configuración seleccionada en pantalla. Aporte las capturas de pantalla cambiando el contenido del fichero .ini y verificando el correcto funcionamiento del programa.	8
B. El control de flujo	9
5. Durante la transferencia de información entre dispositivos, puede ocurrir que la velocidad de procesamiento de uno de ellos sea inferior a la del otro, con lo que se podría perder parte de dicha información si no se utilizase algún mecanismo de arbitraje. Este mecanismo es precisamente el control de flujo. En la interfaz RS-232, existen dos modalidades: el control de flujo hardware y el control de flujo software. . .	9
a. Con el apoyo de la herramienta <i>Hércules</i> describe el proceso de control de flujo por hardware al enviar datos de un equipo a otro. En este caso el control de flujo se realiza mediante las líneas RTS y CTS del RS-232. Si lo consideras necesario solicita el Mini Tester al profesor para confirmar visualmente la activación/desactivación de estas líneas.	9
b. Con el apoyo de la herramienta <i>Hércules</i> describe el proceso de control de flujo por software al enviar datos de un equipo a otro. En este caso el control de flujo se hace mediante las líneas de datos por medio del envío de caracteres predeterminados (<code>XOFF</code> para detener y <code>XON</code> para reanudar la transmisión).	9
C. El control de errores de la interfaz serie	10
6. Realice las modificaciones necesarias en el programa resultante del punto 4 (la configuración del puerto serie se lee del fichero serie.ini) de tal forma que en la recepción se puedan detectar los posibles errores. Consulte “Serial Status” del manual “MSDN - Serial Communications”. Adjunte el proyecto con el código fuente modificado.	10

7.	Debido a que el entorno en el que se desarrollan las prácticas posee una tasa de error muy baja es difícil detectar los posibles errores. Por esta razón, y siempre sobre el código del ejercicio anterior, se pide lo siguiente:	12
a.	Modifique las configuraciones del puerto serie en emisor y receptor para que se puedan producir un errores de paridad (<code>CE_RXPARITY</code>). Especifique y justifique los valores escogidos para la configuración.	12
b.	Modifique las configuraciones del puerto serie en emisor y receptor para que se puedan producir errores de formato (<code>CE_FRAME</code>).Especifique y justifique los valores escogidos para la configuración.	12
c.	Modifique el código para que se pueda obtener en el receptor una condición de línea a 0 ó sin tensión (<code>CE_BREAK</code>). Pista: Revise el API de Windows para el manejo del puerto serie.	12
d.	Modifique el código para que no comience el proceso de recepción hasta que no se haya pulsado la tecla <code>ESC</code> . Especifique los pasos para provocar un error de sobreescritura del buffer de entrada (<code>CE_RXOVER</code>) con el nuevo código.	12
D. Terminal por el puerto serie		13
8.	Una de las aplicaciones prácticas del puerto serie es la de permitir controlar un sistema Unix de forma remota conectado directamente por dicho puerto. Precisamente ésta es la manera más típica que existe para la gestión de muchos elementos de red (p. ej: conmutadores, enrutadores, pasarelas, etc.).	13
a.	Localice e investigue la orden <code>getty</code> para conseguir sacar un terminal por el puerto serie con una velocidad de 9600 baudios y con una emulación para el terminal <code>vt100</code>	13

Índice de figuras

1.	Algunos ejemplos de ejecución del programa con el archivo <code>.ini</code> al lado	8
2.	Configuración utilizada para provocar errores de paridad: Simplemente se escogió diferentes tipos de paridad en ambos terminales (par en uno e impar ene el otro).	12
3.	Configuración utilizada para provocar errores de formato: Se utilizaron diferentes tamaños en ambos terminales tanto de bits de datos como de parada, con el fin de forzar el envío de cadenas de bits de longitud no esperada por el receptor	12
4.	Orden utilizada, tras que el cliente haya cerrado la sesión.	13
5.	Cliente usando <code>putty</code> para conectarse.	13

Bloque A: El cable de módem nulo

1. Construya un cable de módem nulo con el fin de comunicar dos equipos por el puerto serie (recomendada versión completa).

- a) Use el programa *PuTTY* para comprobar dicha comunicación. ¿Qué parámetros se pueden modificar en el puerto serie?. ¿Qué valores son posibles en cada uno de ellos?. Pruebe distintas configuraciones (al menos dos) y describa lo que ocurre al teclear caracteres en una y otra consola. ¿Se puede seleccionar el conjunto de caracteres que el terminal remoto está utilizando?. ¿Para qué sirve esto?.

Parámetros modificables y valores aceptados

Puerto: Cualquier nombre de archivo que represente un puerto de serie.

Pulsos por segundo (baudios): Cualquiera de los valores estándares que enumeramos a continuación:

110, 300¹, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 128000, 153600², 230400, 256000, 460800 y 921600.

Bits de datos de un carácter: 5, 6, 7 u 8 bits.

Bits de stop: 1, 1.5 ó 2.

Paridad: Ninguna, par o impar.

Control de flujo: Ninguno, XON/XOFF o RTS/CTS.

Configuraciones probadas

Configuración por defecto:

9600 baudios, sin paridad, control de flujo XON/XOFF, 8 bits por carácter y un bit de stop, con la que funcionaba sin problemas.

Diferentes pulsos por segundo:

Un lado a 9600 baudios y otro a 14400. Funcionaba también sin ningún problema (probablemente de casualidad).

Diferente paridad:

Un lado con paridad par y otro con paridad impar. Funcionaba bien, probablemente porque *PuTTY* no controlará los errores de paridad, por extraño que parezca.

Diferente tamaño de carácter:

Un lado con 5 bits y otro con 8: Tuvo el efecto esperado, no pudiendo enviar nada valioso entre ambos terminales.

Diferente tamaño de carácter (2):

Un lado con 7 bits y otro con 8: La terminal configurada con 8 bits podía enviar caracteres ASCII a la otra, aunque el octavo bit se perdía.

En la otra dirección se enviaban caracteres no imprimibles, como se podía esperar.

Dato curioso

Para enviar caracteres especiales pudimos usar: `echo "Hola \x00" > /dev/ttyS0`

¹Valor mínimo con el que pudimos transmitir datos

²El valor máximo que nuestra conexión soportaba

2. Modifique la configuración del puerto serie entre 8 y 7 bits de datos

a) *Justifique lo que ocurre al transmitir en ambos casos la letra ñ*

Con 8 bits la ñ se transmite sin problema, ya que la representación de dicho carácter en la codificación que estábamos usando (UTF-8) ocupa 8 bits.

Sin embargo con 7 bits no ocurre lo mismo, el bit más significativo (que resulta ser 1 en dicha codificación) se pierde, haciendo que llegue un carácter diferente al destinatario.

3. Pruebe los programas de ejemplo síncrono y asíncrono para Windows y Linux

a) *Comentar el funcionamiento de los ejemplos facilitados*

Linux

./chat

Envía el texto de forma asíncrona al otro extremo del puerto serie, haciendo que se vea reflejado en el acto en la pantalla.

No se manejan los caracteres de control como el backspace y el carácter nulo, aunque debería ser trivial hacerlo.

./frase

Envía el texto de forma síncrona en bloques de 80 caracteres.

Windows

El funcionamiento de los programas es idéntico al de Linux.

b) *Especifique los parámetros de configuración del puerto serie en cada uno de los ejemplos e indique también la/s línea/s de código en la que se configuran*

Linux

En ambos ejemplos se configura el puerto mediante la función `serie_abrir`, con el puerto directamente sacado de los argumentos, y la velocidad de 9600 baudios.

Dentro de esa función (archivo `serielin/serielin.c:30`) se configura:

- Tamaño de carácter 8 (línea 42).
- Sin paridad (línea 43).
- Tamaño de buffer 1 (línea 46).
- Sin timeout de lectura (línea 47).
- La velocidad la pasada como argumento (50 y 51).
- Otro porrón de implicaciones que se basan en el valor de determinadas constantes (modo no canónico, etc.) (líneas 44 y 45).
- Los parámetros se guardan en la línea 64.

Windows

La configuración del puerto serie se realiza a través de la función `serie_abrir`, localizada en el archivo `seriewin.c` (línea 27)

- Apertura del puerto (línea 32).
- Obtención de la configuración anterior (línea 49).
- Configuración de los baudios (línea 55).
- Configuración del número de bits (línea 56).

- Configuración de la paridad (líneas 59 – 65).
- Configuración de los bits de stop (líneas 68 – 72).
- Hacer que no sea sensible al pin DSR (línea 74).
- Configurar para que se active el DTR cuando se abra el puerto (línea 75).
- Configurar para que no se observe el pin DSR para hacer control de flujo (línea 76).
- Guardado de la configuración (línea 85).
- Configurar los timeouts a 0 (líneas 91 – 104).

c) *Comentar brevemente el API de Windows y Linux para configurar el puerto serie, y cómo se lee y escribe.*

Las APIs tanto de Windows como de Linux son similares en cuanto a que tratan al puerto de serie como un fichero normal, pero a partir de ahí la cosa cambia radicalmente:

La interfaz de Linux (**termios**) es la estándar de POSIX, aunque su diseño puede ser discutible y está ligado al inicio de las comunicaciones entre ordenadores (obtener una terminal en otro ordenador).

La de Windows es una API mucho más genérica, que encaja con el resto del diseño de *Windows NT*.

d) *¿En qué se diferencia el modelo síncrono del asíncrono?*

En el modelo síncrono las lecturas son bloqueantes, mientras que en el modelo asíncrono la lectura se lleva a cabo cuando el estado del puerto de serie cambia (**SIGIO** en Linux, **CommEvent** en Windows), y la ejecución es no lineal.

e) *¿Qué ocurre si cruzamos el programa de ejemplo asíncrono para Windows con el de Linux? Justifique la respuesta.*

En principio debería de funcionar sin problemas (modulo cambios de codificación de caracteres), gracias a la estandarización de la lectura/escritura en los puertos de serie.

4. Cree un fichero de texto con nombre `serie.ini` como el que figura a continuación:

```
[Configuracion]
Puerto=COM1
Velocidad=9600
BitsDatos=7
BitsParada=2
Paridad=Paridad par
```

a) *Modifique y adjunte el programa de ejemplo asíncrono para Windows de forma que cargue los datos de configuración del puerto serie de este fichero. Utilice las funciones de Windows `GetPrivateProfileString` y `GetPrivateProfileInt` para recuperar directamente los valores del archivo de configuración.*

Se migró el código a *Visual Studio* porque el compilador de *DEV-C++* en Windows 10 padecía de errores internos.

De todas formas la migración no fue excesivamente complicada, salvo por ciertos problemas con la configuración del proyecto (*Visual Studio* define la constante **UNICODE** por defecto, lo que cambia el ancho del carácter que usa la API de *Windows*, por ejemplo), o de restricciones del compilador (fuerza determinados castings, el uso de `_getch` vs `getch`, etc).

Los cambios aplicados al archivo son los siguientes (`etc/chat-4.c`):

```
diff --git a/chat.c b/windows-source-fixed.c
index b3cb550..69b84d7 100644
--- a/chat.c
```

```
+++ b/windows-source-fixed.c
@@ -26,6 +26,7 @@
#include <stdio.h>
#include <string.h>
#include <windows.h>
+#include <conio.h>
#include "seriewin.h"

//-----
@@ -50,22 +51,44 @@ int main(int argc, char *argv[])
{
HANDLE hCommPort;

-- //-- Comprueba argumentos del main
if(argc < 2)
{
printf("La sintaxis de este programa es:\n\n");
printf("%s PUERTO\n\n", argv[0]);
printf("\tPUERTO\tNombre de puerto serie:\n");
printf("\t\t\t\t\t\t\tCOM1 Primer puerto serie\n");
printf("\t\t\t\t\t\t\tCOM2 Segundo puerto serie\n");
printf("\tEjemplo:\n");
printf("\t%s COM1\n",argv[0]);
exit(1);
}
+ const char* config_file = ".\\serie.ini";
+ unsigned int baud_rate = GetPrivateProfileInt("Configuracion",
+                                             "Velocidad",
+                                             9600,
+                                             config_file);
+ unsigned int data_bits = GetPrivateProfileInt("Configuracion",
+                                             "BitsDatos",
+                                             8,
+                                             config_file);
+ unsigned int stop_bits = GetPrivateProfileInt("Configuracion",
+                                             "BitsParada",
+                                             2,
+                                             config_file);
+ char port_name[30];
+ unsigned int _len = GetPrivateProfileString("Configuracion",
+                                             "Puerto",
+                                             "COM1",
+                                             port_name,
+                                             sizeof(port_name),
+                                             config_file);
+ char parity[20];
+ unsigned int _parity_len = GetPrivateProfileString("Configuracion",
+                                             "Paridad",
+                                             "Sin paridad",
+                                             parity,
+                                             sizeof(parity),
+                                             config_file);
+ printf("Datos: %d %d %d %s %s\n", baud rate, data bits, stop bits, port name, parity)
```

```

    //-- Abre el puerto serie
-   hCommPort = serie_abrir(argv[1], 9600, 8, 1, "Sin paridad");
-   if( hCommPort == INVALID_HANDLE_VALUE )
+   hCommPort = serie_abrir(port_name, baud_rate, data_bits, stop_bits, parity);
+   if (hCommPort == INVALID_HANDLE_VALUE)
    {
        perror("ERROR: No se puede acceder al puerto serie.");
        exit(2);
@@ -74,7 +97,7 @@ int main(int argc, char *argv[])
    //-- Crea el hilo encargado de la lectura de datos del puerto serie
    HANDLE hHiloLectura;
    DWORD idHiloLectura;
-   hHiloLectura = CreateThread(NULL,0,(void *)HiloLectura,&hCommPort,0,&idHiloLectura);
+   hHiloLectura = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)HiloLectura, &hCommPort, 0,
↪ &idHiloLectura);
    if (hHiloLectura == NULL)
    {
        perror("ERROR: No se puede crear el hilo de lectura");
@@ -87,7 +110,7 @@ int main(int argc, char *argv[])
    char c;
    do
    {
-       c = getch();
+       c = _getch();
        if (c != ESC) {
            serie_escribir(hCommPort, &c, sizeof(c));
        }

```

- b) Verifique su correcto funcionamiento mostrando la configuración seleccionada en pantalla. Aporte las capturas de pantalla cambiando el contenido del fichero .ini y verificando el correcto funcionamiento del programa.

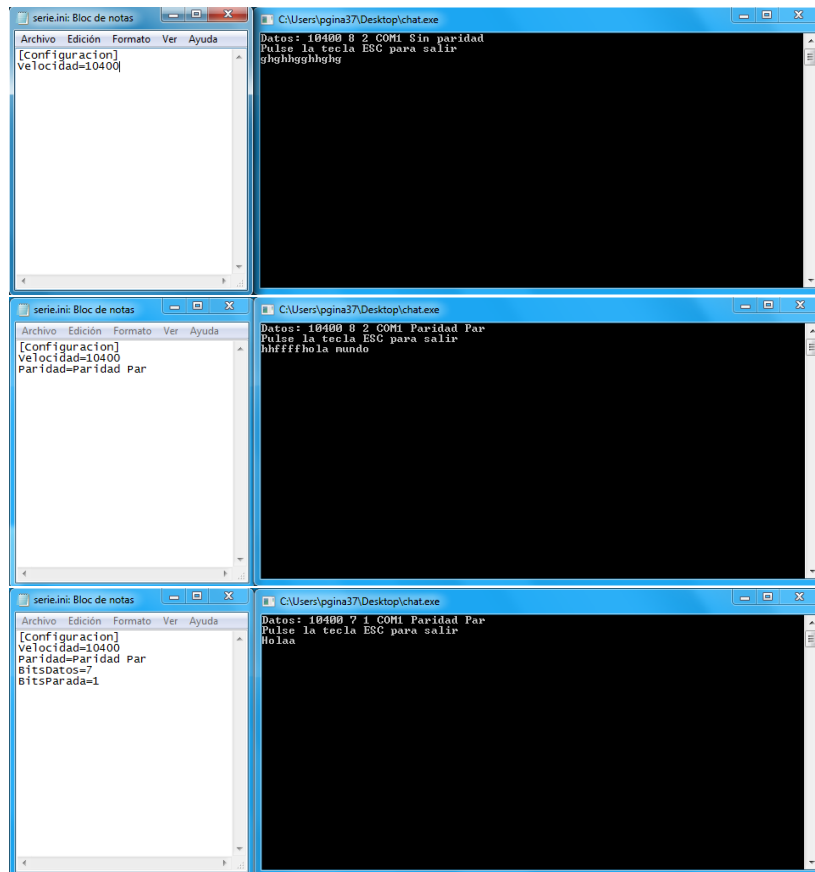


Figura 1: Algunos ejemplos de ejecución del programa con el archivo .ini al lado

Bloque B: El control de flujo

5. Durante la transferencia de información entre dispositivos, puede ocurrir que la velocidad de procesamiento de uno de ellos sea inferior a la del otro, con lo que se podría perder parte de dicha información si no se utilizase algún mecanismo de arbitraje. Este mecanismo es precisamente el control de flujo. En la interfaz RS-232, existen dos modalidades: el control de flujo hardware y el control de flujo software.

- a) *Con el apoyo de la herramienta Hércules describe el proceso de control de flujo por hardware al enviar datos de un equipo a otro. En este caso el control de flujo se realiza mediante las líneas RTS y CTS del RS-232. Si lo consideras necesario solicita el Mini Tester al profesor para confirmar visualmente la activación/desactivación de estas líneas.*

Cuando el puerto está cerrado, el pin RTS está en negativo.

Cuando el puerto del otro ordenador se intenta conectar, el CTS pasa a positivo, ya que el otro ordenador a activado su RTS (que está cruzado), lo que indica que el otro terminal está listo.

Si conectamos el puerto, nuestro RTS pasa también a positivo (y por lo tanto el CTS del otro terminal).

Cuando ambos pines en ambos dispositivos están en positivo, podemos transmitir datos.

Si uno de los dos terminales usa RTS/CTS y el otro no, el primer terminal no enviará datos, pero sí de trata de recibirlos (pone el pin RTS a positivo, pero no recibe la señal de CTS, por lo que no envía). Sin embargo el otro terminal sí envía, y los datos llegan al primero.

- b) *Con el apoyo de la herramienta Hércules describe el proceso de control de flujo por software al enviar datos de un equipo a otro. En este caso el control de flujo se hace mediante las líneas de datos por medio del envío de caracteres predeterminados (XOFF para detener y XON para reanudar la transmisión).*

Con XON y XOFF, tanto el pin RTS como el CTS están en negativo (ya que no se usan). El control de flujo se realiza por software (cuando se recibe el carácter XOFF se deja de transmitir, hasta que se vuelva a recibir el carácter XON).

Bloque C: El control de errores de la interfaz serie

- Realice las modificaciones necesarias en el programa resultante del punto 4 (la configuración del puerto serie se lee del fichero serie.ini) de tal forma que en la recepción se puedan detectar los posibles errores. Consulte “Serial Status” del manual “MSDN - Serial Communications”. Adjunte el proyecto con el código fuente modificado.

Sólo se ha usado `printf` para detectar el error, ya que el manejo del mismo no se especificaba en el enunciado.

Nótese también que se ha usado `GetCommMask` y `SetCommMask` para evitar modificar el archivo `seriewin.c`.

Los cambios aplicados con respecto al otro ejercicio son los siguientes (`etc/chat-6.c`):

```
diff --git a/chat-4.c b/chat-6.c
index 16c061a..8e2b5cf 100644
--- a/chat-4.c
+++ b/chat-6.c
@@ -94,6 +94,17 @@ int main(int argc, char *argv[])
     exit(2);
 }

+ DWORD mask;
+ if (!GetCommMask(hCommPort, &mask)) {
+     perror("GetCommMask");
+     exit(3);
+ }
+
+ if (!SetCommMask(hCommPort, mask | EV_ERR)) {
+     perror("SetCommMask");
+     exit(4);
+ }
+
+ //-- Crea el hilo encargado de la lectura de datos del puerto serie
+ HANDLE hHiloLectura;
+ DWORD idHiloLectura;
@@ -101,7 +112,7 @@ int main(int argc, char *argv[])
     if (hHiloLectura == NULL)
     {
         perror("ERROR: No se puede crear el hilo de lectura");
-        exit(3);
+        exit(5);
     }

     printf("Pulse la tecla ESC para salir\n");
@@ -139,8 +150,7 @@ DWORD HiloLectura(LPDWORD lpParam)
 {
     DWORD dwEvtMask;

-    OVERLAPPED ov;
-    memset(&ov, 0, sizeof(ov));
+    OVERLAPPED ov = { 0 };
     ov.hEvent = CreateEvent(0, TRUE, 0, 0);

     //-- Recoge el manejador del puerto serie
@@ -148,12 +158,32 @@ DWORD HiloLectura(LPDWORD lpParam)

     char szDatos[MAXCHARS + 1];
```

```

+   DWORD dwErrors;
+
+   //-- Bucle infinito de lectura
+   while (TRUE)
+   {
-       if (!WaitCommEvent(hCommPort, &dwEvtMask, &ov))
-       {
-           //-- perror("ERROR: WaitCommEvent. Esperando evento síncrono");
+       if (!WaitCommEvent(hCommPort, &dwEvtMask, &ov)) {}
+
+       if (!ClearCommError(hCommPort, &dwErrors, NULL)) {
+           perror("ClearCommError");
+           continue;
+       }
+
+       if (dwErrors & CE_FRAME) {
+           printf("Frame error\n");
+       }
+       if (dwErrors & CE_RXPARITY) {
+           printf("Parity error\n");
+       }
+       if (dwErrors & CE_OVERRUN) {
+           printf("Overrun\n");
+       }
+       if (dwErrors & CE_BREAK) {
+           printf("CE_BREAK\n");
+       }
+       if (dwErrors & CE_RXOVER) {
+           printf("CE_RXOVER\n");
+       }
+
+       //-- Espera a que haya algo que leer
@@ -162,7 +192,7 @@ DWORD HiloLectura(LPDWORD lpParam)
    DWORD dwMask;
    if (GetCommMask(hCommPort, &dwMask))
    {
-        if (dwMask == EV_TXEMPTY)
+        if (dwMask & EV_TXEMPTY)
        {
            ResetEvent(ov.hEvent);
            continue;
@@ -172,6 +202,8 @@ DWORD HiloLectura(LPDWORD lpParam)
        //-- Lee los datos del puerto serie
        if (serie_leer(hCommPort, szDatos, sizeof(szDatos)) > 0)
            printf("%s", szDatos);
+
+        else
            printf("serie_leer returned 0");

        //-- Reinicia el estado del evento
        ResetEvent(ov.hEvent);

```

7. Debido a que el entorno en el que se desarrollan las prácticas posee una tasa de error muy baja es difícil detectar los posibles errores. Por esta razón, y siempre sobre el código del ejercicio anterior, se pide lo siguiente:

- a) *Modifique las configuraciones del puerto serie en emisor y receptor para que se puedan producir un errores de paridad (CE_RXPARITY). Especifique y justifique los valores escogidos para la configuración.*

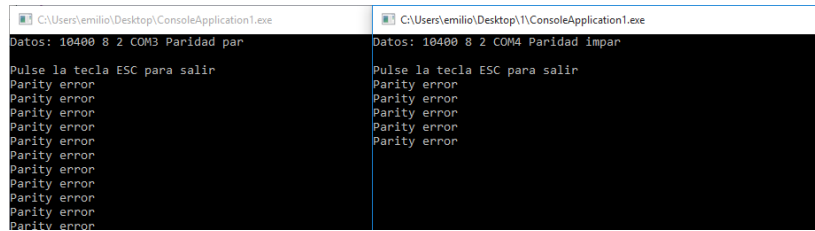


Figura 2: Configuración utilizada para provocar errores de paridad: Simplemente se escogió diferentes tipos de paridad en ambos terminales (par en uno e impar en el otro).

- b) *Modifique las configuraciones del puerto serie en emisor y receptor para que se puedan producir errores de formato (CE_FRAME). Especifique y justifique los valores escogidos para la configuración.*

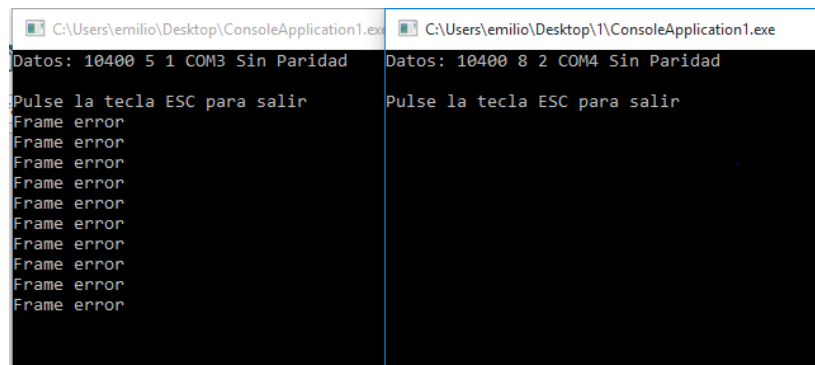


Figura 3: Configuración utilizada para provocar errores de formato: Se utilizaron diferentes tamaños en ambos terminales tanto de bits de datos como de parada, con el fin de forzar el envío de cadenas de bits de longitud no esperada por el receptor

- c) *Modifique el código para que se pueda obtener en el receptor una condición de línea a 0 ó sin tensión (CE_BREAK). Pista: Revise el API de Windows para el manejo del puerto serie.*

Nuestro código originalmente ya podía detectar los CE_BREAK, así que no ha habido necesidad de modificarlo de nuevo.

- d) *Modifique el código para que no comience el proceso de recepción hasta que no se haya pulsado la tecla ESC. Especifique los pasos para provocar un error de sobrescritura del buffer de entrada (CE_RXOVER) con el nuevo código.*

Bloque D: Terminal por el puerto serie

8. Una de las aplicaciones prácticas del puerto serie es la de permitir controlar un sistema Unix de forma remota conectado directamente por dicho puerto. Precisamente ésta es la manera más típica que existe para la gestión de muchos elementos de red (p. ej: conmutadores, enrutadores, pasarelas, etc.).

- a) Localice e investigue la orden `getty` para conseguir sacar un terminal por el puerto serie con una velocidad de 9600 baudios y con una emulación para el terminal `vt100`.

Al usar `getty` se abre el puerto serie y se permite crear una terminal remota desde un cliente.

La orden ejecutada, como se puede ver en las capturas de pantalla del directorio `screenshots`, fue:

```
getty ttyS0 9600 vt100
```

Nótese que el argumento `vt100` es completamente innecesario, ya que es por defecto.

La consola se queda leyendo de la entrada estándar indefinidamente, hasta que el cliente remoto cierra la sesión.

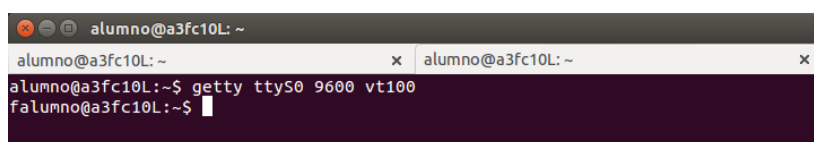


Figura 4: Orden utilizada, tras que el cliente haya cerrado la sesión.

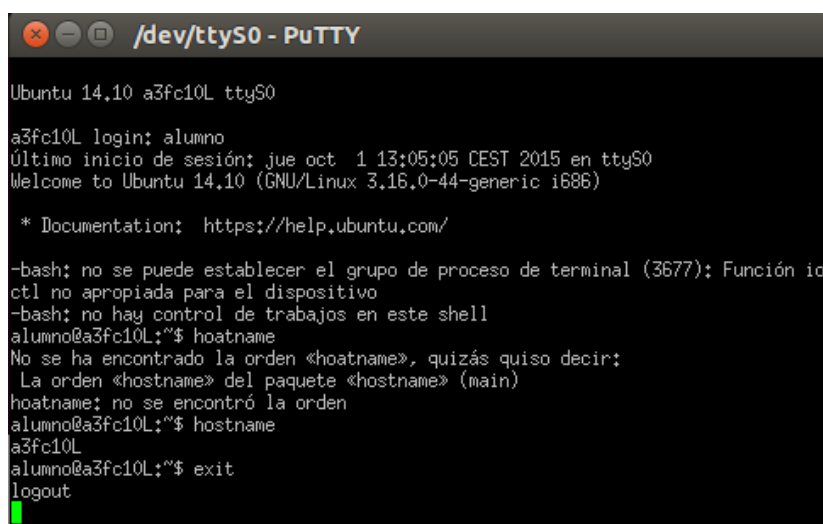


Figura 5: Cliente usando putty para conectarse.