Practica 4

Periféricos e Interfaces Industriales

Departament d'Informàtica de Sistemes i Computadors

DISCA

Comunicación de dos PC's a través del puerto serie

OBJETIVOS

La práctica consistirá en la conexión de dos PC's a través del puerto serie, realizando el conexionado NULL-Módem para puerto serie RS-232, y programación en Visual C++ de dos programas para la comunicación. Para la programación del puerto serie utilizaremos las rutinas de alto nivel que nos proporciona el API Win32.

MATERIAL NECESARIO

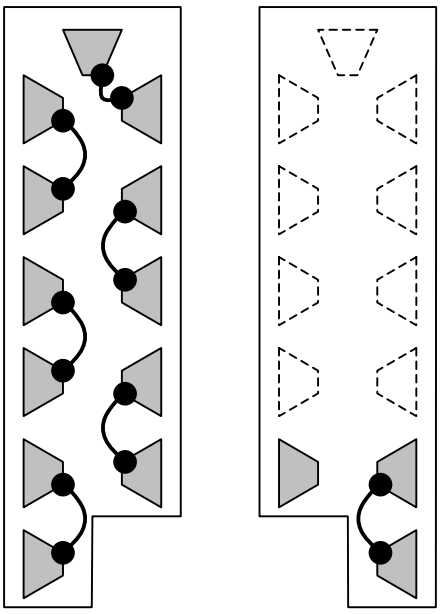
Es conveniente traer un disquete para poder grabar los programas que se realicen y también las transparencias del tema 9: Comunicaciones serie.

TRABAJO PREVIO

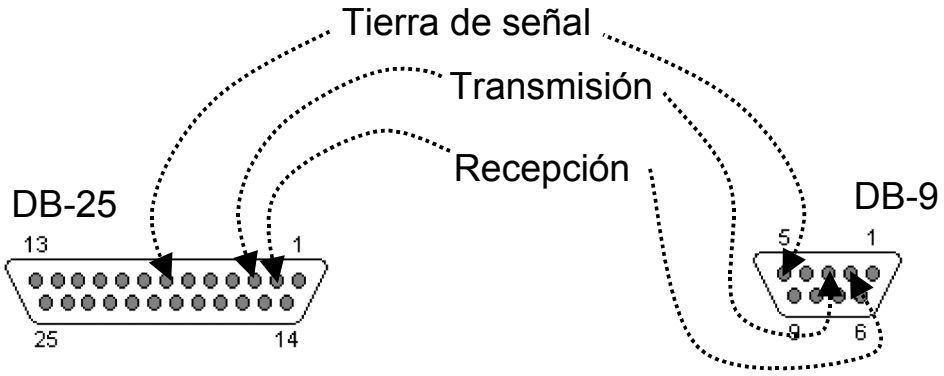
Recomendamos refrescar las ideas sobre C y Visual C++ y mirar la comunicación serie como se detalla en las transparencias que podéis consultar en ftp://kona2.alc.upv.es/pub/PEI/Tranparecies/T6_centronics.pdf.

CONEXIONADO

Los PC's nuevos del laboratorio de informática del DISCA se encuentran conectados mediante un cable serie nulo, tal como hemos visto en las transparencias de clase y tal como se detalla en la figura de la página siguiente. Están conectados de la siguiente forma:

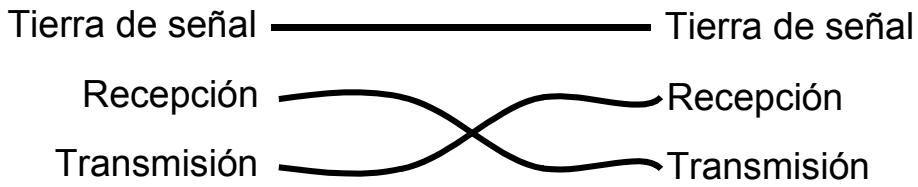


Colocarse de forma que ocupéis dos ordenadores conectados por grupo (siempre que sea posible) para que podáis realizar las pruebas mas cómodamente.
De todas formas habrá un par de ordenadores libres para aquellos que no dispongáis de dos ordenadores.



Pin 2: Transmisión de datos
Pin 3: Recepción de datos
Pin 5 (DB-9) o Pin 7 (DB-25):
Tierra de señal

Conexionado:



COMUNICACIÓN SERIE A ALTO NIVEL

Una vez visto en clase como se realiza la configuración y comunicación a través del puerto serie utilizando programación a bajo nivel (directamente sobre la UART) y a nivel medio (mediante la BIOS), vamos a tratar la comunicación a utilizando un lenguaje de alto nivel (Visual C++) y las funciones que la API (Aplication Program Interface) de Windows 9X y NT (API Win32) nos proporciona.

Para ello deberemos ver como se estructuran las aplicaciones en Win32, que funciones disponemos para el acceso al puerto y que posibilidades tenemos.

Ya que la programación y comunicación del puerto serie suele ser bastante delicada (a todos los niveles) y dado que no se pretende que seáis unos expertos programadores, se os orientará sobre todos los pasos a realizar comentados y solo tendréis que realizar el esqueleto del programa y colocar el código necesario.

Por último, se supone que tenéis los mínimos conceptos necesarios de Visual C++ para desarrollar una pequeña aplicación basada en una caja de dialogo, tal como hemos visto en clase.

API WIN32 PARA PUERTO SERIE

Es importante destacar que los sistemas operativos (en mayor o menor medida) MS Windows 9X y NT son sistemas operativos multitarea. Es decir que simultáneamente se están ejecutando en el ordenador diversos programas; no en la CPU que solo puede ejecutar uno a la vez. Esto significa que todos los programas comparten la misma CPU y por tanto, mientras se está ejecutando un proceso o tarea no se esta ejecutando realmente otro.

Por ello nuestros programas no deben ejecutarse "inútilmente" para malgastar tiempo de CPU: No debemos utilizar algoritmos de POOLING o consulta para realizar acciones, por ejemplo comprobar si hemos recibido algún dato por el puerto serie. Sin embargo directamente desde Win32 tampoco podemos utilizar el método de Interrupciones puesto que estas las gobierna el sistema operativo y no son "utilizables directamente" como ocurría en MS-DOS o en los microcontroladores.

SOLUCIÓN: Las funciones del sistema operativo son bloqueantes de forma que "desalojan" la CPU mientras no se pueden realizar (porque nadie nos ha enviado nada). Es decir, si ejecutamos la rutina de leer del puerto serie (o de donde sea), podemos hacer que el programa se pare hasta que esta lectura del dato se realice:

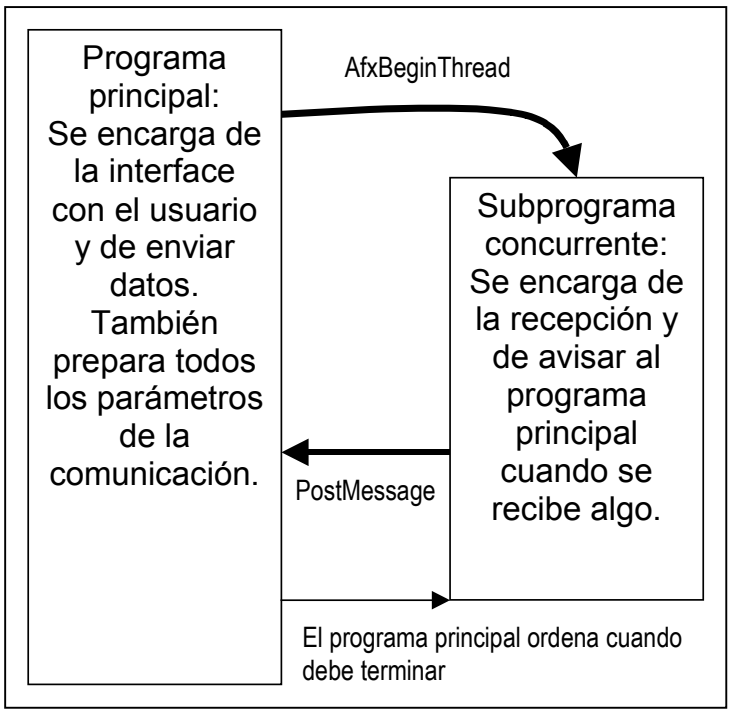
```
.....programa.....
Leer puerto serie
.... ya tenemos el dato leído, y si no había ninguno para leer se
... habrá esperado hasta tenerlo...
```

PROBLEMA: Tal como probablemente veríais en la práctica anterior, si el programa se queda bloqueado esperando recibir algo, no puede realizar ninguna acción más: es decir no puede enviar nada tampoco!.

Y aquí es donde la cosa se complica: la solución pasa por arrancar un segundo *thread* (o hilo, o hebra).

Que es un *thread*: Es una rutina, programa, etc. que se ejecuta al mismo tiempo que el programa principal (como si fueran varios procesos). Con ello, podemos arrancar un *thread* que se encargue de la recepción de los datos del puerto serie, mientras que el programa principal se encarga de manejar la interfaz con el usuario y el envío de datos por el puerto.

NO UTILIZAREMOS ESTE MÉTODO DEBIDO A SU MAYOR COMPLEJIDAD.



Para simplificar el programa realizaremos envíos y recepciones que bloquean al programa principal. Esto no supondrá ningún problema para el caso del envío, ya que los bits se envían aunque nadie este recibiéndolos, pero si que será un problema para la recepción, ya que cuando tengamos que recibir X bits, hasta que no se reciben, no se sale de la función.

LLAMADAS DE WIN32 RELACIONADAS CON LA COMUNICACIÓN SERIE:

CreateFile()	Abre el puerto serie. Igual que con ficheros de disco.
SetCommState()	Configura las comunicaciones.
GetCommState()	Lee la configuración actual de las comunicaciones.
ReadFile()	Lee del puerto serie. Igual que con ficheros de disco.
WriteFile()	Escribe en el `puerto. Igual que con ficheros de disco.
CloseHandle()	Cierra el puerto serie para que lo puedan usar otros programas.
SetCommMask()	Establece que tipo de información despierta a WaitCommEvent...
WaitCommEvent()	Espera que ocurra un evento: recepción, envio, errores, ...
SetupComm()	Establece el tamaño de los buffers.
PurgeComm()	Limpia los buffers y posibles errores ocurridos.
SetCommTimeouts()	Establece los tiempos máximos de espera para las operaciones.
ClearCommError()	Limpia errores producidos.

Solo vamos a utilizar las llamadas indispensables para realizar nuestra comunicación:

- *CreateFile()*: Permite la creación, apertura, escritura de ficheros, puertos de comunicación, tuberías (PIPES), etc.
- *ReadFile()*: Lee del puerto (fichero o tubería) abierto con CreateFile(), y bloquea o no al que ejecuta la operación según el modo con que se haya abierto. En nuestro caso si que la bloqueará.
- *WriteFie()*: Ídem pero para escritura.
- *GetCommState()* y *SetCommState()*: para configurar los parámetros de la transmisión serie: Número de bits, bits de stop, paridad, velocidad, etc...
- *CloseHandle()*: Cierra el puerto (fichero o tubería) para que lo puedan utilizar otros programas. Hay que tener en cuenta, que mientras tengamos abierto el puerto, ningún otro programa podrá acceder a el.

Se recomienda consultar la ayuda del Visual C++ para una descripción completa y detallada de cada una de ellas con todos los parámetros que pueden aceptar.

EJERCICIO

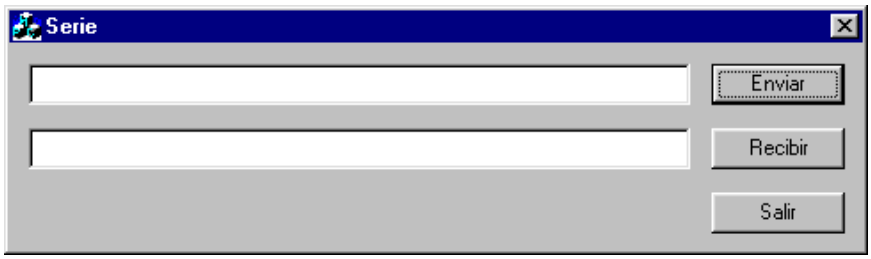
Se propone implementar un programa parcialmente escrito a continuación para realizar una comunicación simple NULL – módem entre los dos equipos a base de caracteres primero y para enviar ficheros de texto a continuación.

La comunicación se debe realizar con los parámetros:

8 bits de datos, 1 bit de stop, paridad par y 9600 baudios.

También deberéis colocar el puerto serie adecuado según el cableado esté sobre COM1 o COM2 (En el este caso los PC están conectados por COM1).

El programa podría tener un aspecto parecido a este (a gusto de consumidor):



En el que le indicamos el fichero a transmitir o recibir y pulsamos la tecla de enviar o recibir para efectuar la operación. Ya que no estamos implementando ningún protocolo, sino simplemente enviando datos y recibiéndolos por el extremo opuesto, el *receptor se deberá arrancar antes*, ya que en otro caso, el emisor enviara los datos aunque no lleguen a nadie.

Para que el receptor sepa cuando se ha terminado el fichero podemos utilizar 2 métodos:

- Enviar como primeros 2 (o 4) bytes la longitud del fichero, tal como vimos en el puerto paralelo.
- Enviar un carácter especial para indicar el fin del fichero. En este caso optaremos por este método pero esto nos limita el tipo de ficheros a enviar a tipo texto, porque en un fichero binario, el carácter especial podría aparecer en el interior del fichero y terminar la comunicación antes del final (se debería utilizar la técnica del bit stuffing). Además tiene la pega de que tendremos que analizar uno por uno los bytes recibidos.

Para el **envío** el código ha utilizar debería tener el siguiente aspecto:

- **APERTURA DEL PUERTO:**

```
HANDLE m_hComm;  //Manejador del puerto de comunicaciones
m_hComm=CreateFile( "COM1", GENERIC_WRITE | GENERIC_READ,
//Abrimos para enviar (write) o recibir (read)
                    FILE_SHARE_READ,
                    NULL,
                    //Sin seguridad
                    OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL,
                    //Necesario para puertos serie
                    NULL);
//Aquí podemos indicarle si las operaciones son bloqueantes

//Comprobación de si se han producido errores
if(m_hComm==(HANDLE)-1) {
   AfxMessageBox("Error abriendo el puerto serie COM1");
   return;
}
```

- **CONFIGURACION DEL PUERTO (9600, 8 BITS, PAR Y 1 STOP):**

```
   DCB dcb; //Estructura de datos configuracion del puerto
   dcb.DCBlength = sizeof( DCB ) ;
   GetCommState( m_hComm, &dcb );
   //Recuperamos la configuracion actual
   dcb.BaudRate = CBR_9600;  //Velocidad
   dcb.ByteSize = 8;         //num. bits
   dcb.Parity = EVENPARITY;  //Paridad par
   dcb.fParity = TRUE ;      //Habilitamos la paridad
   dcb.StopBits = ONESTOPBIT; //1 Bit de stop

   dcb.fBinary = TRUE ;
   SetCommState( m_hComm, &dcb ) ;
```

- **ABRIMOS EL FICHERO INDICADO PARA LEERLO:**

```
   Cfile fichero;
   if(fichero.Open(m_Enviar, CFile::modeRead)==0) {
        AfxMessageBox("Error abriendo el fichero");
        CloseHandle(m_hComm);
        return;
   }
```

- **LEEMOS EL FICHERO EN UN BUFFER DE DATOS Y VAMOS ENVIANDO:**

```
unsigned long enviat=0, enviando=0, leidos;
unsigned char buffer[512];          //Buffer
do {
   leidos=fichero.Read(buffer, 512);
   //Leemos bytes en el buffer
   enviat=0, enviando=0;
   //enviat: numero de bytes enviados
   //enviando: numero de bytes enviados en esta instruccion
   while(enviat!=leidos) {
       //hasta que no hayamos enviado todo el buffer...
        WriteFile(m_hComm,  //Handle del puerto
         &(buffer[enviat]), //Dirección del buffer
             leidos-enviat,
            //leidos-enviat: lo que queda por enviar
            &enviando, NULL);
            //Lo que realmente se ha logrado enviar

        enviat+=enviando;
   }
} while(leidos==512);
//Cuando lo logremos llenar el buffer con el fichero
// significa que se ha acabado el fichero
char eof=0;
WriteFile(m_hComm, &eof, 1, &enviando, NULL);
//Enviamos carácter de final de fichero (0, NULL)
CloseHandle(m_hComm);
//Cierre del puerto serie
```

Para la **recepción** el código ha utilizar debería tener el siguiente aspecto:

- **APERTURA DEL PUERTO:**

```
IDEM A LA TRANSMISIÓN
```

- **CONFIGURACION DEL PUERTO (9600, 8 BITS, PAR Y 1 STOP):**

```
IDEM A LA TRANSMISIÓN
```

- **ABRIMOS EL FICHERO INDICADO PARA ESCRIBIRLO:**

```
   Cfile fichero;
   if(fichero.Open(m_Recibir, CFile::modeCreate |
                                  CFile::modeWrite)==0) {
        AfxMessageBox("Error abriendo el fichero");
        CloseHandle(m_hComm);
        return;
   }
```

- **"LEEMOS" DEL PUERTO SERIE Y ESCRIBIMOS EN ESE FICHERO:**

```
unsigned long rebut=0, recibiendo=0;
char cadena[512];           //Buffer de recepción
int i=0;
do {
   rebut=0;                 //Indice de recepción
   do {
        ReadFile(m_hComm,   //HANDLE del puerto
         &(cadena[rebut]), //Dirección del buffer
         1,                //Numero de bytes a leer
         &recibiendo,      //Numero de bytes leidos
         NULL);
        if(cadena[rebut]==0) break;
             //Terminamos cuando se recibe un 0
        rebut+=recibiendo;
             //Incrementamos la cuenta de bytes leidos
   } while(rebut<512);
        //Cuando tenemos el buffer lleno o hemos terminado
        // salimos y escribimos los bytes recibidos en
        // el fichero
   fichero.Write(cadena, rebut);
        //Repetimos mientras el buffer se haya llenado
        // del todo (quedan bytes por recibir).
} while(rebut==512);
        //Cerramos el puerto serie
CloseHandle(m_hComm);
```

Esta rutina tiene la desventaja de que debemos mirar cada byte que se recibe para comprobar si es el último byte del fichero (el 0) ya que si intentamos leer cuando ya se ha terminado el envio de datos, la función *ReadFile* se quedará bloqueada y con ella nuestro programa.

Pasos a desarrollar:

1. Implementar el programa descrito y probar que funciona.
2. Probar que este protocolo no es válido para enviar ficheros binarios (por ejemplo una imagen).
3. Cambiar el protocolo para que se envien 4 bytes iniciales con el tamaño del fichero que se va ha enviar y para que el receptor no tenga que analizar 1 a 1 los bytes que se envian.
4. Comprobar que este nuevo protocolo sirve para todo tipo de ficheros.
5. En caso de querer introducir mejoras y tener tiempo, añadir la posibilidad al usuario de cambiar la velocidad de transmisión y el puerto (tal como aparece en la figura del programa).