LnxComm

Linux Communication

Fernando Pujaico Rivera

LnxComm : Linux Communication

por Fernando Pujaico Rivera

<fernando.pujaico.rivera (en) gmail.com>

Tabla de contenidos

Capítulo 1. Introducción

La biblioteca "Linux Communication" (LnxComm) está diseñada para brindar un apoyo a los programadores que estén relacionados con el diseño y construcción de hardware. LnxComm nos permite crear una conexión con el puerto serie mediante unas pocas lineas de código.

Otra de las ventajas de esta biblioteca es que nos permite crear programas que podrán ser compilados en sistemas operativos GNU-LINUX y WINDOWS brindando así mayor portabilidad a nuestros programas.

La biblioteca está completamente desarrollada en Lenguaje C.

Capítulo 2. Descripción de funciones

2.1. Previamente

Las funciones cumplen las mismas características tanto como para sistemas operativos GNU-LINUX y WINDOWS.

2.2. Open_Port

Función de lectura del puerto.

```
HANDLE Open_Port(char COMx[]);
```

Abre el puerto serie, recibe como parámetro una cadena con el nombre del puerto y devuelve una variable de tipo *HANDLE* que es el manejador del puerto.

```
COMx[]:  Es una cadena que contiene el nombre del puerto a abrir, ejemplo.
```

En Windows:

```
"COM1"        , "COM2"        , "COM3"        , "COM4"        , ...
```

En Gnu-Linux:

```
"/dev/ttyS0"  , "/dev/ttyS1"  , "/dev/ttyS2"  , "/dev/ttyS3"  , ...
"/dev/ttyUSB0", "/dev/ttyUSB1", "/dev/ttyUSB2", "/dev/ttyUSB3", ...
"/dev/ttyACM0", "/dev/ttyACM1", "/dev/ttyACM2", "/dev/ttyACM3", ...
```

Retorna: El manejador de Puerto Abierto (variable tipo *HANDLE*). En caso de error devuelve *INVALID_HANDLE_VALUE*.

2.3. Get_Configure_Port

Devuelve configuración actual del puerto serie.

```
DCB Get_Configure_Port(HANDLE fd);
```

Esta función devuelve un variable de tipo *DCB* con la configuración actual del puerto serie ,la función recibe un parámetro de tipo *HANDLE* que es el manejador devuelto por la función Open_port.

```
fd     :  Es el manejador del puerto.
```

Retorna: Una estructura *DCB* con una copia de la configuración actual del puerto serie y carga la variable *ERROR_CONFIGURE_PORT* con *FALSE*, en caso de error carga la variable *ERROR_CONFIGURE_PORT* con *TRUE*.

2.4. Configure_Port

Establece la configuración del puerto serie.

```
DCB Configure_Port(    HANDLE          fd,
                       unsigned int    BaudRate,
                       char            CharParity[]);
```

Esta función configura el puerto serie con los parámetros *fd*,*BaudRate* y *CharParity*.

```
fd        : Es el manejador del puerto serie devuelto por Open_port.
BaudRate  : Es la velocidad del puerto serie. (B115200, B19200, B9600, ...)
CharParity: Indica el número de bits de la transmisión. ("8N1","7E1","7O1","7S1")
```

Retorna: Una estructura *DCB* con una copia de la configuración actual del puerto serie y carga la variable *ERROR_CONFIGURE_PORT* con *FALSE*, en caso de error carga la variable *ERROR_CONFIGURE_PORT* con *TRUE*.

2.5. Set_Configure_Port

Establece la configuración del puerto serie.

```
int Set_Configure_Port(        HANDLE  fd,
                               DCB     PortDCB);
```

Restituye/establece la configuración del puerto serie, los parámetros serán pasados mediante una variable tipo *DCB*.

```
fd     : Es el manejador del puerto devuelto por Open_port.
newtio : Es una variable DCB con la configuración del puerto, generalmente
         se usa la devuelta por la función Get_Configure_Port
```

Retorna: *TRUE* si todo fue bien o *FALSE* si hubo algún error.

2.6. Write_Port

Escribe un bloque de datos tipo *char* en el puerto serie.

```
long Write_Port(       HANDLE  fd,
                       char    Data[],
                       int     SizeData);
```

Escribe los *SizeData* primeros caracteres de *Data*.


Se debe escoger un *SizeData* menor o igual que la longitud de *Data*.

```
fd  : Es el manejador del puerto devuelto por Open_port.
Data  : Es el dato a mandar.
SizeData: Es el número de bytes que se quieren escribir.
```

Retorna : En caso de éxito devuelve el número de bytes escritos (cero indica que no se ha escrito nada). En GNU-LINUX en caso de error devuelve -1.

2.7. Read_Port

Recibe un bloque de datos en el puerto serie.

```
long Read_Port(         HANDLE  fd,
                        char    *Data,
                        int     SizeData);
```

Lee los *SizeData* primeros caracteres del puerto y lo carga en *Data*.

 Se debe escoger un *SizeData* menor o igual que la longitud de *Data*.

```
fd  : Es el manejador del puerto devuelto por Open_port.
Data  : Es la variable en donde se reciben los datos.
SizeData: Es el número de bytes que se desea recibir.
```

Retorna : En caso de éxito devuelve el número de bytes leídos (cero indica que no se ha leído nada). En GNU-LINUX en caso de error devuelve -1.

2.8. Gets_Port

Recibe una cadena de caracteres tipo *char* por el puerto serie.

```
long Gets_Port(         HANDLE  fd,
                        char    *Data,
                        int     SizeData);
```

Recibe datos por el puerto, lee hasta encontrar un 0x0A,0x0D o hasta completar *SizeData* caracteres. Los datos son guardados en la variable *Data*

 Se debe escoger un *SizeData* menor o igual que la longitud de *Data*.

```
fd      : Es el manejador del puerto devuelto por Open_port.
Data    : Es la variable en donde se reciben los datos.
SizeData: Es el máximo número de bytes que se desea recibir.
```

Retorna : El número de caracteres recibidos, estos números serán siempre mayores o iguales a cero.

2.9. Getc_Port

Recibe un caracter por el puerto serie.

```
long Getc_Port(        HANDLE  fd,
                       char    *Data);
```

Recibe un único caracter por el puerto y es cargado en la variable *Data* de tamaño 1 byte.

```
fd     : Es el manejador del puerto devuelto por Open_port.
Data   : Es la variable en donde se reciben los datos(1 Byte).
```

Retorna : En caso de éxito devuelve el número de bytes leídos. En GNU-LINUX en caso de error devuelve -1.

2.10. Kbhit_Port

Indica el número de bytes en el buffer de entrada del puerto serie.

```
int Kbhit_Port( HANDLE fd);
```

Recibe como parámetro el manejador del puerto.

```
fd     : Es el manejador del puerto devuelto por Open_port.
```

Retorna: El número de caracteres en el buffer de entrada.

2.11. Close_Port

Cierra el puerto serie.

```
int Close_Port( HANDLE fd);
```

Recibe la variable *fd* y cierra el puerto serie.

```
fd     : Es el manejador del puerto devuelto por Open_port.
```

Retorna: *TRUE* si se ha cerrado el puerto y *FALSE* en el caso contrario.

2.12. Set_Hands_Haking

Configura el control de flujo en el puerto serie.

```
int Set_Hands_Haking(  HANDLE  fd,
                       int     FlowControl);
```

Recibe como variables el manejador del puerto serie y el tipo de control de flujo.

```
fd         : Es el manejador del puerto devuelto por Open_port.
FlowControl: Es un número entero que indica el tipo de control de flujo.
               0    Ninguno
               1    RTSCTS
               2    XonXoff
               3    DTRDSR
```

Retorna : *TRUE* si todo fue bien y *FALSE* si no lo fue.

2.13. Set_BaudRate

Configura la velocidad en baudios del puerto serie.

```
int Set_BaudRate(      HANDLE fd,
                       unsigned int BaudRate);
```

Recibe como datos el manejador del puerto y la velocidad en baudios del mismo.

```
fd      : Es el manejador del puerto devuelto por Open_port.
BaudRate: Es la velocidad del puerto, los valores pueden ser.
              B2400
              B9600
              B19200
              B115200
       Para mas datos vea el archivo baudios.h .
```

Retorna: *TRUE* si todo fue bien y *FALSE* si no lo fue.

2.14. Set_Time

Configura temporizador para las funciones de lectura y escritura en el puerto serie.

```
int Set_Time(  HANDLE          fd,
               unsigned int    Time);
```

Recibe como variables, el manejador del puerto y el máximo tiempo entre bytes en milisegundos (ms)

```
fd     : Es el manejador del puerto devuelto por Open_port.
Time   : Multiplicador, para el tamaño total del TimeOut en la
         lectura y escritura de datos.

       TimeOut = (100 * Time * numero_de_bytes_en_la_lectura) ms
```

Retorna: *TRUE* si todo fue bien y *FALSE* si no lo fue.

2.15. IO_Blocking

Escoge entre el modo bloqueante y no bloqueante en lectura de datos en el puerto serie.

```
int IO_Blocking(       HANDLE fd,
                       int Modo);
```

La función recibe como parámetro el manejador del puerto y *TRUE* si se quiere una lectura de datos bloqueante o *FALSE* si no.

```
fd     : Es el manejador del puerto devuelto por Open_port.
Modo   : TRUE : Modo bloqueante.
         FALSE: Modo no bloqueante.
```

Retorna: *TRUE* si todo fue bien y *FALSE* si no lo fue.

2.16. Clean_Buffer

Termina las operaciones de lectura y escritura pendientes y limpia las colas de recepción y de transmisión en el puerto serie.

```
int Clean_Buffer(      HANDLE fd);
```

La función recibe como parámetro el manejador del puerto.

```
fd     : Es el manejador del puerto devuelto por Open_port.
```

Retorna: *TRUE* si todo fue bien y *FALSE* si no lo fue.

2.17. Create_Thread_Port

Crea una función (hilo) que se ejecuta cuando existan caracteres en el buffer de entrada del puerto serie.

```
pthread_t Create_Thread_Port(  HANDLE *fd);
```

Recibe como parametro el manejador del puerto. y devuelve una variable de tipo *pthread_t*

```
fd     : Es el manejador del puerto devuelto por Open_port.
```

Retorna: El manejador del hilo creado.

Para poder usar la función Create_Thread_Port primero se debe de habilitar escribiendo lo siguiente: *#define ENABLE_SERIAL_PORT_EVENT*, luego se debe de escribir el código de la función *SERIAL_PORT_EVENT(HANDLE *hPort)*

```
#define ENABLE_SERIAL_PORT_EVENT

#include "com/serial.h"

void SERIAL_PORT_EVENT(        HANDLE *hPort)
```

```
{
        // Código de ejemplo aquí
        // char   Data[16];
        // Read_Port(*hPort,Data,15);
        // Data[15]=0;
        // printf("%s",Data);
}
```

La función *SERIAL_PORT_EVENT* recibe como parámetro un puntero de tipo HANDLE que es el manejador del puerto devuelto por Open_port.

Capítulo 3. Ejemplos

Para escribir tus programas puedes escoger cuatro métodos bloqueante, no-bloqueante, time-out y evento.

3.1. Previamente

La cabecera cambia según el sistema operativo.

```
Linux:

#define __LINUX_COM__
#include "com/serial.h"

Windows:

#define __WINDOWS_COM__
#include "com/serial.h"
```

El uso de las comillas dobles (*"com/serial.h"*) indica que la carpeta *com* se encuentra en la misma carpeta del archivo de código fuente que la invoca, osea si se tiene un archivo *ejemplo.c* que usa *"com/serial.h"*, la carpeta *com* se debe de encontrar en la misma carpeta de *ejemplo.c*.

La función Open_Port también cambiara de argumento según el sistema operativo.

```
Linux:

"/dev/ttyS0"  , "/dev/ttyS1"  , ...
"/dev/ttyUSB0", "/dev/ttyUSB1", ...
"/dev/ttyACM0", "/dev/ttyACM1", ...

Windows:

"COM1", "COM2, "COM3", ...
```

Si se esta usando linux como sistema operativo la compilación de los programas que se realicen se hará de la siguiente manera:

```
gcc -o archivo archivo.c
```

En el caso de que se esté creando un hilo se deberá compilar de la siguiente manera.

```
gcc -o archivo archivo.c -lpthread
```

3.2. Bloqueante

Aquí (../../bloqueante.c) se tiene un ejemplo de un programa bloqueante.

```
#define __LINUX_COM__                    // #define __WINDOWS_COM__
```

```
#include "com/serial.h"

int main()
{
    HANDLE fd;
    DCB OldConf;
    char cad[16]="Enviando Texto";
    int n;

    fd=Open_Port("/dev/ttyS0");         // Abre el puerto serie.
                                        // fd=Open_Port("COM1");

    OldConf=Get_Configure_Port(fd);     // Guardo la configuración del puerto.

    Configure_Port(fd,B115200,"8N1");   // Configuro el puerto serie.

                                        // Bloqueante por defecto, pero también
                                        // se puede usar:
                                        // IO_Blocking(fd,TRUE);

    n=Write_Port(fd,cad,16);            // Escribo en el puerto serie.

    while(Kbhit_Port(fd)<16);           // Espero a leer hasta que se tengan
                                        // 16 bytes en el buffer de entrada.

    n=Read_Port(fd,cad,16);             // Leo el puerto serie.
    printf("%s",cad);                   // Muestro los datos.

    Set_Configure_Port(fd,OldConf);     // Restituyo la antigua configuración
                                        // del puerto.

    Close_Port(fd);                     // Cierro el puerto serie.

    printf("\nPresione ENTER para terminar\n");
    getchar();

    return 0;
}
```

3.3. No-Bloqueante

Aquí (../../nobloqueante.c) se tiene un ejemplo de un programa no-bloqueante.

```
#define __WINDOWS_COM__                 // #define __LINUX_COM__

#include "com/serial.h"

int main()
{
    HANDLE fd;
    DCB OldConf;
```

```
    char cad[16]="Enviando Texto";
    int n;

    fd=Open_Port("COM1");               // Abre el puerto serie.
                                        // fd=Open_Port("/dev/ttyS0");

    OldConf=Get_Configure_Port(fd);     // Guardo la configuración del puerto.

    Configure_Port(fd,B115200,"8N1");   // Configuro el puerto serie.

    IO_Blocking(fd,FALSE);              // Seleccionamos lectura no bloqueante.

    n=Write_Port(fd,cad,16);            // Escribo en el puerto serie.

    while(Kbhit_Port(fd)<16);           // Espero a leer hasta que se tengan
                                        // 16 bytes en el buffer de entrada.

    n=Read_Port(fd,cad,16);             // Leo el puerto serie.
    printf("%s",cad);                   // Muestro loas datos.


    Set_Configure_Port(fd,OldConf);     // Restituyo la antigua configuración
                                        // del puerto.

    Close_Port(fd);                     // Cierro el puerto serie.

    printf("\nPresione ENTER para terminar\n");
    getchar();

    return 0;
}
```

3.4. Timeout

Aquí (../../timeout.c) se tiene un ejemplo de un programa con Time-Out.

```
#define __WINDOWS_COM__                 // #define __LINUX_COM__

#include "com/serial.h"

int main()
{
    HANDLE fd;
    DCB OldConf;
    char cad[16]="X";
    int n,TIME=2,i;

    fd=Open_Port("COM1");               // Abre el puerto serie.
                                        // fd=Open_Port("/dev/ttyS0");

    OldConf=Get_Configure_Port(fd);     // Guardo la configuración del puerto.
```

```
    Configure_Port(fd,B19200,"8N1");    // Configuro el puerto serie.

    Set_Time(fd,TIME);                  // Time-Out entre caracteres es TIME*0.1.

    n=Write_Port(fd,cad,1);             // Escribo en el puerto serie.

    n=Gets_Port(fd,cad,16);             // Leo el puerto serie.
    printf("%s",cad);                   // Muestro la cadena.

    Set_Configure_Port(fd,OldConf);     // Restituyo la antigua configuración
                                        // del puerto.
    Close_Port(fd);                     // Cierro el puerto serie.

    printf("\nPresione ENTER para terminar\n");
    getchar();

    return 0;
}
```

3.5. Evento

Aquí (../../evento.c) se tiene un ejemplo de un programa con evento.

```
#define __WINDOWS_COM__                 // #define __LINUX_COM__
#define ENABLE_SERIAL_PORT_EVENT

#include "com/serial.h"

int numero=0;
void SERIAL_PORT_EVENT(HANDLE * hPort)
{
    char c;
    Getc_Port(*hPort,& c);
    printf("[%d]=%c\n",numero,c);
    numero++;
}

int main()
{
    HANDLE fd;
    DCB OldConf;
    char cad[16]="Enviando Texto\n";
    int n;

    fd=Open_Port("COM1");               // Abre el puerto serie.
                                        // fd=Open_Port("/dev/ttyS0");

    OldConf=Get_Configure_Port(fd);     // Guardo la configuración del puerto.

    Configure_Port(fd,B115200,"8N1");   // Configuro el puerto serie.
```

```
    IO_Blocking(fd,TRUE);               // Bloqueante por defecto, pero también
                                        // se puede usar:
                                        // IO_Blocking(fd,TRUE);

    n=Write_Port(fd,cad,16);            // Escribo en el puerto serie.

    Create_Thread_Port(& fd);        // Creo un hilo y le paso el manejador.

    while(TRUE);

    Set_Configure_Port(fd,OldConf);     // Restituyo la antigua configuración
                                        // del puerto.

    Close_Port(fd);                     // Cierro el puerto serie.

    printf("\nPresione ENTER para terminar\n");
    getchar();

    return 0;
}
```

Capítulo 4. Referencias

- http://lnxcomm.sourceforge.net (http://lnxcomm.sf.net)
- http://zsoluciones.com
- http://winapi.conclase.net
- http://google.com