Mise en oeuvre du panneau lumineux Mc Crypte 590996

Thierry Vaira <tvaira@free.fr>

Table des matières

Site : tvaira.free.fr

Mise en oeuvre du panneau lumineux Mc Crypte 590996

Ressources :

- www.conradpro.fr
- eduscol.education.fr/sti/
- genelaix.free.fr
- Communication_protocol_LED_Displ_Board.pdf

GNU/Linux

Prise en charge

Il est conseillé de lire ou de s'aider de ce document pour comprendre la prise en charge d'un périphérique USB sous Linux.

Avant :

```
$ lsusb > lsusb-avant.txt
$ lsmod > lsmod-avant.txt
$ dmesg > dmesg-avant.txt
```

Brancher maintenant le panneau lumineux sur un port USB.

Après :

```
$ lsusb > lsusb-apres.txt
$ lsmod > lsmod-apres.txt
$ dmesg > dmesg-apres.txt
```

Analyse :

```
$ diff lsusb-avant.txt lsusb-apres.txt
> Bus 001 Device 008: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x Composite Device

$ diff dmesg-avant.txt dmesg-apres.txt
> usb 1-1.2.1: new full-speed USB device number 8 using ehci-pci
> usb 1-1.2.1: New USB device found, idVendor=10c4, idProduct=ea60
> usb 1-1.2.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
> usb 1-1.2.1: Product: CP2102 USB to UART Bridge Controller
> usb 1-1.2.1: Manufacturer: Silicon Labs
> usb 1-1.2.1: SerialNumber: 0001
> usbcore: registered new interface driver usbserial
> usbcore: registered new interface driver usbserial_generic
> usbserial: USB Serial support registered for generic
> usbcore: registered new interface driver cp210x
> usbserial: USB Serial support registered for cp210x
> cp210x 1-1.2.1:1.0: cp210x converter detected
> usb 1-1.2.1: reset full-speed USB device number 8 using ehci-pci
> usb 1-1.2.1: cp210x converter now attached to ttyUSB0

$ diff lsmod-avant.txt lsmod-apres.txt
> cp210x                 17431  0
> usbserial              37161  1 cp210x

$ modinfo cp210x
filename:       /lib/modules/3.8.0-44-generic/kernel/drivers/usb/serial/cp210x.ko
license:        GPL
description:    Silicon Labs CP210x RS232 serial adaptor driver
srcversion:     11C809808F2C0D6EA6DB810
...
alias:          usb:v10C4pEA60d*dc*dsc*dp*ic*isc*ip*in*
...
depends:        usbserial
intree:         Y
vermagic:       3.8.0-44-generic SMP mod_unload modversions

$ ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 mars  18 09:41 /dev/ttyUSB0
```

- le périphérique USB a été détecté physiquement par le noyau (cf. lsusb et dmesg)
- c'est un "CP210x Composite Device" de Cygnal Integrated Products (idVendor=10c4, idProduct=ea60)
- ce périphérique peut être pris en charge par le module chargeable `cp210x` (cf. lsmod et modinfo)
- le pilote de périphérique (*driver*) associé à ce matériel (`cp210x`) a été chargé par le système ainsi que le module `usbserial` (cf. dmesg et lsmod)
- le périphérique sera accessible par le fichier spécial `ttyUSB0` dans `/dev` (cf. dmesg et ls)

Attention : le périphérique ne sera accessible en lecture/écriture que par l'utilisateur propriétaire `root` et les membres du groupe `dialout`.

Par défaut, le système applique la politique des droits d'accès `rw-rw----` pour ce type de périphérique USB :

```
$ ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 mars  18 09:41 /dev/ttyUSB0
```

Ici, le périphérique sera accessible en lecture/écriture par l'utilisateur propriétaire `root` et les membres du groupe `dialout`. Tous les autres (*other*) utilisateurs auront aucun accès.

Pour modifier cette situation, vous avez les possibilités suivantes :

- changer les droits manuellement (à chaque fois !) : `$ sudo chmod 666 /dev/ttyUSB0`
- ajouter l'utilisateur demandeur (ici toto) dans le groupe `dialout` : `$ sudo adduser toto dialout`

Pour une gestion automatique, il faudra passer par `udev` qui est maintenant le service qui prend en charge le répertoire `/dev`.

```
$ sudo vim /etc/udev/rules.d/51-ttyusb.rules
SUBSYSTEMS=="usb", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea60", MODE="0666"
```

Il est aussi possible de créer un lien symbolique au choix :


```
$ sudo vim /etc/udev/rules.d/51-ttyusb.rules
SUBSYSTEMS=="usb", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea60", MODE="0666", SYMLINK+="panneau"
```

Programmation en C sous Linux

Lire la documentation sur le protocole de communication du panneau : Communication_protocol_LED_Displ_Board.pdf.

```c
/*
 * Test du panneau Mc Crypte-590996 (sous Linux)
 */

#include <stdio.h>
#include <errno.h>
#include <termios.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/signal.h>
#include <fcntl.h>

#define DEBUG_PANNEAU

#define PORT            "/dev/ttyUSB0"

#define LG_MAX_TRAME    128
#define LG_MAX          16
#define LG_REPONSE      4 // au maximum 4 caractères pour NACK

// cf. man ascii
#define NUL             0x00 // caractère NUL (c'est aussi le code du fin de chaîne)
#define ACK             0x06 // accusé réception positif
#define NACK            0x15 // accusé réception négatif

#define DELAI           1000000 // en micro secondes

// Lire: http://ftp.lip6.fr/pub/linux/french/echo-linux/html/ports-series/ports_series.html
int ouvrirPort(char *nomPort)
{
    int fd = -1;
    struct termios  termios_p;

    #ifdef DEBUG_PANNEAU
    fprintf(stderr, "Ouverture du port %s\n", nomPort);
    #endif

    // cf. man 2 open
    if (( fd=open(nomPort, O_RDWR|O_NONBLOCK )) == -1 )
    {
            perror("open");
            return fd;
    }

    // cf. man tcgetattr
    tcgetattr(fd, &termios_p);

    // configuration du port série : 9600 bits/s, 8 bits, pas de parité
    // remarque : les caractères BREAK et ceux qui comportent une erreur de parité sont ignorés
    termios_p.c_iflag = IGNBRK | IGNPAR;
    // rien de particulier à faire pour l'envoi des caractères
    termios_p.c_oflag = 0;
    // 9600 bits/s, 8 bits, pas de parité
```



```
    termios_p.c_cflag = B9600 | CS8;
    termios_p.c_cflag &= ~PARENB;

    // pas d'écho des caractères reçus
    termios_p.c_lflag = ~ECHO;
    // spécifie le nombre de caractéres que doit contenir le tampon pour être accessible à la lecture
    // En général, on fixe cette valeur à 1
    termios_p.c_cc[VMIN] = 1;
    // spécifie en dixièmes de seconde le temps au bout duquel un caractère devient accessible,
    // même si le tampon ne contient pas [VMIN] caractères
    // Une valeur de 0 représente un temps infini.
    termios_p.c_cc[VTIME] = 0;

    // cf. man tcsetattr
    tcsetattr(fd, TCSANOW, &termios_p);

    // cf. man 2 fcntl
    // mode bloquant ?
    //fcntl(fd, F_SETFL, fcntl(fd,F_GETFL)&~O_NONBLOCK);

    return fd;
}

void fermerPort(int fd)
{
   // cf. man 2 close
   close(fd);
}

int envoyer(int fd, char *trame, int nb)
{
    int retour = -1;

    if(fd > 0)
    {
        // cf. man 2 write
        retour = write(fd, trame, nb);

        #ifdef DEBUG_PANNEAU
        //debug : affichage
        fprintf(stderr, "-> envoyer (%d/%d) : ", nb, retour);
        //fprintf(stderr, "trame : ");
        /*int i;
        for(i=0;i<nb;i++)
        {
            fprintf(stderr, "0x%02X ", *(trame+i));
        }
        fprintf(stderr, "\n");*/
        fprintf(stderr, "%s\n", trame);
        #endif
        if (retour == -1)
        {
            perror("write");
        }
    }
    else
    {
        #ifdef DEBUG_PANNEAU
        //debug : affichage
        fprintf(stderr, "-> envoyer (%d) : ERREUR port !\n", nb);
        fprintf(stderr, "trame : ");
        /*int i;
        for(i=0;i<nb;i++)
        {
            fprintf(stderr, "0x%02X ", *(trame+i));
        }
```



```
        fprintf(stderr, "\n");*/
        fprintf(stderr, "%s\n", trame);
        #endif
        retour = fd;
    }

    return retour;
}

int recevoir(int fd, char *donnees, int nb)
{
    int retour;
    char donnee;
    int lus = 0;

    if(fd > 0 && donnees != (char *)NULL)
    {
        if(nb > 0)
        {
            for(lus=0;lus<nb;lus++)
            {
                // cf. man 2 read
                retour = read(fd, &donnee, 1);
                if(retour > 0)
                        *(donnees+lus) = donnee;
                else
                {
                    /*if (retour == -1)
                    {
                        perror("read");
                    }*/
                    break;
                }
            }
            if(nb == lus && nb > 1)
                *(donnees+lus) = 0x00; //fin de chaine
            retour = lus;
            #ifdef DEBUG_PANNEAU
            int i;
            fprintf(stderr, "<- recevoir (%d/%d) : ", nb, lus);
            //fprintf(stderr, "trame : ");
            for(i=0;i<lus;i++)
                fprintf(stderr, "0x%02X ", *(donnees+i));
            fprintf(stderr, "\n");
            #endif
        }
        else
        {

        }
    }
    else retour = fd;

    return retour;
}

unsigned char calculerChecksum(char *trame)
{
   unsigned char checksum = 0;
   int i = 0;

   #ifdef DEBUG_PANNEAU
   printf("data packet :\t");
   for(i=0;i<strlen(trame);i++)
      printf("0x%02X ", trame[i]);
   printf("\n");
```



```
   #endif

   for(i=0;i<strlen(trame);i++)
      checksum ^= trame[i];

   #ifdef DEBUG_PANNEAU
   printf("checksum :\t0x%02X\n", checksum);
   #endif

   return checksum;
}

/* Programme principal */
int main()
{
    char nomPort[LG_MAX] = { PORT };
    int fd = -1;
    //Exemple de trame : "<ID01><L1><PA><FE><MA><WC><FE>message1F<E>"
    char trame[LG_MAX_TRAME] = { NUL };
    char protocole[LG_MAX_TRAME] = "<L1><PA><FE><MA><WC><FE>";
    char message[LG_MAX] = "BTS-SN"; // attention : longueur max du message égale à LG_MAX-1
    char reponse[LG_MAX] = { NUL };
    unsigned char checksum;
    int retour;

    // 0. on ajoute le message dans l'en-tête du protocole
    sprintf(protocole, "%s%s", protocole, message);

    // 1. on calcule le checksum de la trame
    checksum = calculerChecksum(protocole);

    // 2. on fabrique la trame
    sprintf(trame, "<ID01>%s%02X<E>", protocole, checksum);

    // 3. on transfère la trame

    // 3.1 on ouvre le port
    fd = ouvrirPort(nomPort);
    if(fd == -1)
    {
        fprintf(stderr, "Erreur ouverture !\n");
        //return fd;
    }

    // 3.2 on envoie la trame
    retour = envoyer(fd, &trame[0], strlen(trame));
    if(retour == -1)
    {
        fprintf(stderr, "Erreur transmission !\n");
        // ... ?
    }

    usleep(DELAI);

    // 3.3 on réceptionne l'acquittement
    retour = recevoir(fd, reponse, LG_REPONSE);
    if(retour < 1)
    {
        fprintf(stderr, "Erreur réception !\n");
        // ... ?
    }
    else
    {
        printf("Réponse : %s\n", reponse);
    }
```



```
    // 3.4 on ferme le port
    fermerPort(fd);

    return 0;
}
```

On fabrique l'exécutable :

```
$ gcc -O2 test-panneau-linux.c
```

On teste :

```
$ ./a.out
data packet :   0x3C 0x4C 0x31 0x3E 0x3C 0x50 0x41 0x3E 0x3C 0x46 0x45 0x3E 0x3C 0x4D 0x41 0x3E
                0x3C 0x57 0x43 0x3E 0x3C 0x46 0x45 0x3E 0x6D 0x65 0x73 0x73 0x61 0x67 0x65
checksum :  0x1F
Ouverture du port /dev/ttyUSB0
-> envoyer (42/42) : <ID01><L1><PA><FE><MA><WC><FE>message1F<E>
<- recevoir (4/3) : 0x41 0x43 0x4B
Réponse : ACK
```

Dans le cas d'un mauvais checksum, on obtient :

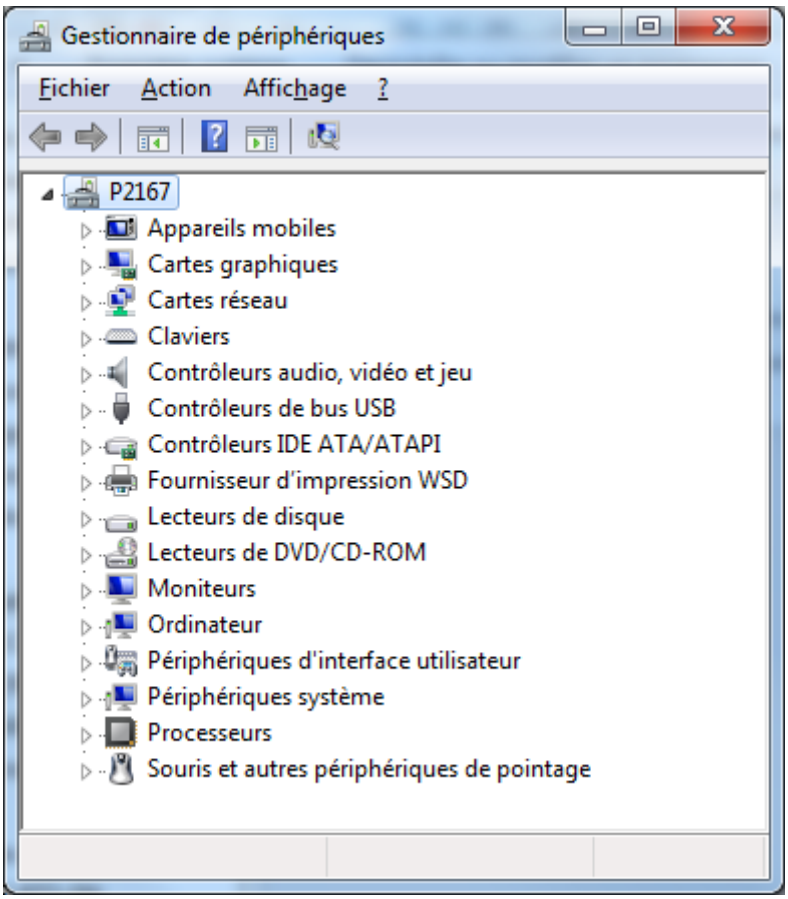
```
$ ./a.out
Ouverture du port /dev/ttyUSB0
-> envoyer (42/42) : <ID01><L1><PA><FE><MA><WC><FE>message00<E>
<- recevoir (4/4) : 0x4E 0x41 0x43 0x4B
Réponse : NACK
```

Code source : test-panneau-linux.c

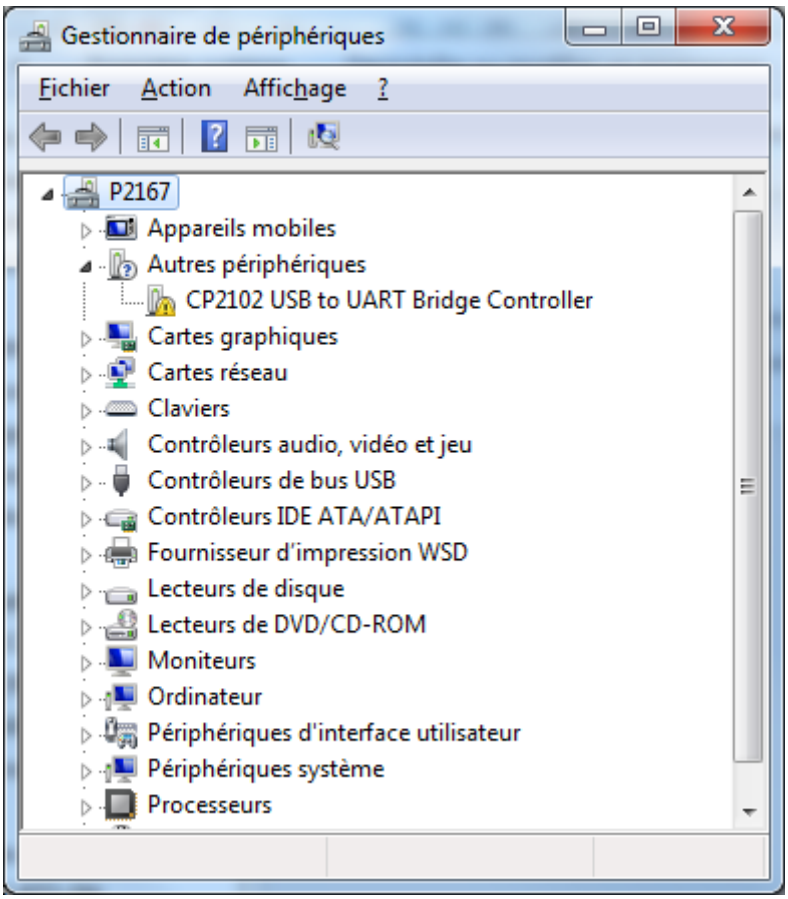
Windows

Prise en charge

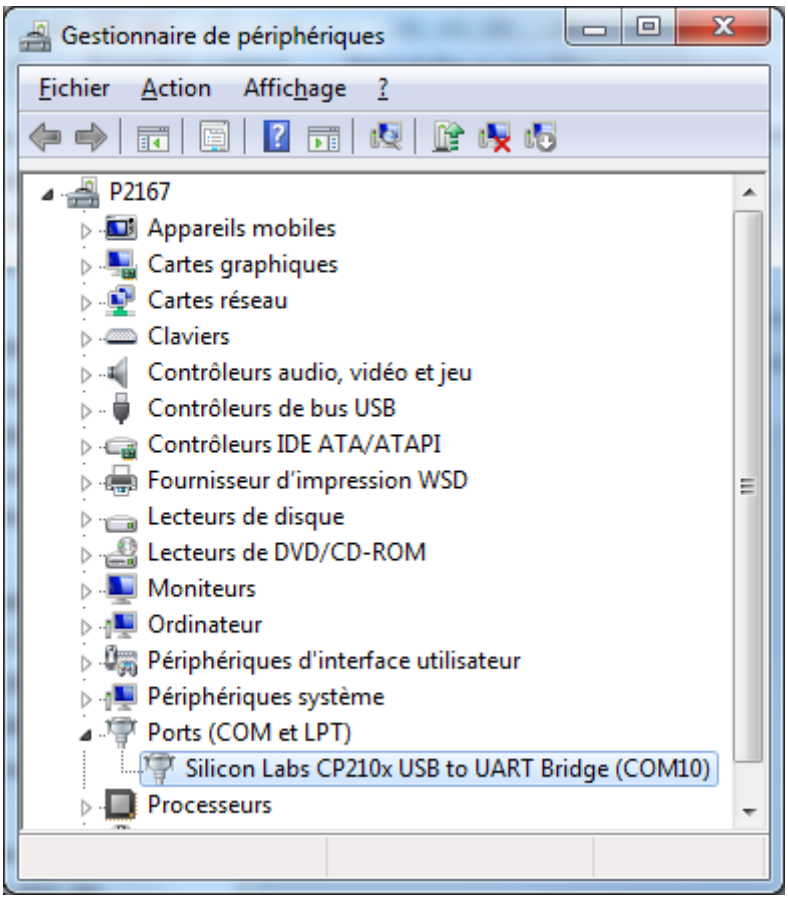
- Démarrer le "gestionnaire de périphériques" pour vérifier la prise en charge du périphérique :
  - Raccourci clavier : "Windows + R" -> Commande "Exécuter"
  - `DEVMGMT.MSC` -> Gestionnaire de Périphériques



– **Brancher maintenant votre périphérique USB.**



– Puis, le périphérique USB devient accessible via un port série virtuel `COM10`



Programmation en C sous Windows

Lire la documentation sur le protocole de communication du panneau : Communication_protocol_LED_Displ_Board.pdf.

Le programme a été testé avec Dev-C++.

```c
/*
 * Test du panneau Mc Crypte-590996 (sous Windows)
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>

#define DEBUG_PANNEAU

#define PORT            "COM10" // cf. gestionnaire de périphériques

#define LG_MAX_TRAME    128
#define LG_MAX          16
#define LG_REPONSE      4 // au minimum 3 caractères pour ACK

// cf. code ascii
#define NUL             0x00 // caractère NUL (c'est aussi le code du fin de chaîne)
#define ACK             0x06 // accusé réception positif
#define NACK            0x15 // accusé réception négatif

#define DELAI           1 // en secondes

// Lire: http://ftp.lip6.fr/pub/linux/french/echo-linux/html/ports-series/ports_series.html
HANDLE ouvrirPort(char *nomPort)
{
```



```
HANDLE hPort = INVALID_HANDLE_VALUE; // Handle sur le port série
DCB old_dcb; // anciens parametres du port série
DCB dcb; // parametres du port série
COMMTIMEOUTS comout = {0}; // timeout du port serie ici = MODE BLOQUANT
COMMTIMEOUTS oldTimeouts; // ancien timeout du port série
char nomPeripherique[LG_MAX] = { PORT };

// \\.\COM10
sprintf(nomPeripherique, "\\\\.\\%s", nomPort);
// Lire https://msdn.microsoft.com/en-us/library/windows/desktop/aa363858%28v=vs.85%29.aspx
hPort = CreateFile(
                    "\\\\.\\COM10",
                    GENERIC_READ | GENERIC_WRITE,
                    0,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL);

if( hPort == INVALID_HANDLE_VALUE )
{
    fprintf(stderr, "Erreur d'ouverture du peripherique %s !\n", nomPeripherique);
    return hPort;
}

/* Lecture des parametres courants  */
GetCommState(hPort, &dcb);
old_dcb = dcb; // sauvegarde l'ancienne configuration

/* Liaison a 9600 bps, 8 bits de donnees, pas de parite, lecture possible */
dcb.BaudRate = CBR_9600;
dcb.ByteSize = 8;
dcb.StopBits = ONESTOPBIT;
dcb.Parity = NOPARITY;
dcb.fBinary = TRUE;
/* pas de control de flux */
dcb.fOutxCtsFlow = FALSE;
dcb.fOutxDsrFlow = FALSE;

/* Sauvegarde des nouveaux parametres */
if( !SetCommState(hPort, &dcb) )
{
  fprintf(stderr, "Impossible de configurer le port %s !", nomPort);
  CloseHandle(hPort);
  return hPort;
}
SetupComm(hPort, 2048, 1024);
GetCommTimeouts(hPort, &oldTimeouts);
// MODE NON BLOQUANT (timeout)
// Specify time-out between charactor for receiving.
comout.ReadIntervalTimeout = 100;
// Specify value that is multiplied
// by the requested number of bytes to be read.
comout.ReadTotalTimeoutMultiplier = 1;
// Specify value is added to the product of the
// ReadTotalTimeoutMultiplier member
comout.ReadTotalTimeoutConstant = 0;
// Specify value that is multiplied
// by the requested number of bytes to be sent.
//comout.WriteTotalTimeoutMultiplier = 3;
// Specify value is added to the product of the
// WriteTotalTimeoutMultiplier member
//comout.WriteTotalTimeoutConstant = 2;
// set the time-out parameter into device control.
SetCommTimeouts(hPort, &comout);
```



```
    return hPort;
}

void fermerPort(HANDLE hPort)
{
    // Lire https://msdn.microsoft.com/en-us/library/windows/desktop/ms724211%28v=vs.85%29.aspx
    CloseHandle(hPort);
}

BOOL envoyer(HANDLE hPort, char *trame, int nb)
{
    BOOL retour = FALSE;
    DWORD nNumberOfBytesToWrite = nb;
    DWORD ecrits;

    if(hPort > 0)
    {
        // Lire https://msdn.microsoft.com/en-us/library/windows/desktop/aa365747%28v=vs.85%29.aspx
        retour = WriteFile(hPort, trame, nNumberOfBytesToWrite, &ecrits, NULL);

        #ifdef DEBUG_PANNEAU
        //debug : affichage
        fprintf(stderr, "-> envoyer (%d/%d) : ", nb, ecrits);
        //fprintf(stderr, "trame : ");
        /*int i;
        for(i=0;i<nb;i++)
        {
            fprintf(stderr, "0x%02X ", *(trame+i));
        }
        fprintf(stderr, "\n");*/
        fprintf(stderr, "%s\n", trame);
        #endif
        if (retour == FALSE)
        {
            fprintf(stderr, "Erreur écriture port !");
        }
    }
    else
    {
        #ifdef DEBUG_PANNEAU
        //debug : affichage
        fprintf(stderr, "-> envoyer (%d) : ERREUR port !\n", nb);
        fprintf(stderr, "trame : ");
        /*int i;
        for(i=0;i<nb;i++)
        {
            fprintf(stderr, "0x%02X ", *(trame+i));
        }
        fprintf(stderr, "\n");*/
        fprintf(stderr, "%s\n", trame);
        #endif
    }

    return retour;
}

// Lire https://msdn.microsoft.com/en-us/library/windows/desktop/aa365467%28v=vs.85%29.aspx
BOOL recevoir(HANDLE hPort, char *donnees, int nb)
{
    char donnee;
    int n;
    DWORD lus = 0;
    BOOL retour = FALSE;

    if(hPort > 0 && donnees != (char *)NULL)
    {
```

```
        if(nb > 0)
        {
            for(n=0;n<nb;n++)
            {
                // ATTENTION au mode bloquant !
                retour = ReadFile(hPort, &donnee, 1, &lus, NULL);
                if(retour == TRUE)
                {
                    if(donnee != 0)
                    {
                        *(donnees+n) = donnee;
                        fprintf(stderr, "donnee : 0x%02X (%d)\n", *(donnees+n), n);
                    }
                    else n--;
                }
                else
                {
                    break;
                }
            }
            *(donnees+n) = 0x00; //fin de chaine
            #ifdef DEBUG_PANNEAU
            int i;
            fprintf(stderr, "<- recevoir (%d/%d) : ", nb, n);
            //fprintf(stderr, "trame : ");
            for(i=0;i<nb;i++)
                fprintf(stderr, "0x%02X ", *(donnees+i));
            fprintf(stderr, "\n");
            #endif
        }
        else
        {

        }
    }

    return retour;
}

unsigned char calculerChecksum(char *trame)
{
   unsigned char checksum = 0;
   int i = 0;

   #ifdef DEBUG_PANNEAU
   printf("data packet :\t");
   for(i=0;i<strlen(trame);i++)
      printf("0x%02X ", trame[i]);
   printf("\n");
   #endif

   for(i=0;i<strlen(trame);i++)
      checksum ^= trame[i];

   #ifdef DEBUG_PANNEAU
   printf("checksum :\t0x%02X\n", checksum);
   #endif

   return checksum;
}

/* Programme principal */
int main()
{
    char nomPort[LG_MAX] = { PORT };
    HANDLE hPort = INVALID_HANDLE_VALUE; // Handle sur le port série
```



```
    //Exemple de trame : "<ID01><L1><PA><FE><MA><WC><FE>message1F<E>"
    char trame[LG_MAX_TRAME] = { NUL };
    char protocole[LG_MAX_TRAME] = "<L1><PA><FE><MA><WC><FE>";
    char message[LG_MAX] = "message"; // attention : longueur max du message égale à LG_MAX-1
    char reponse[LG_MAX] = { NUL };
    unsigned char checksum;
    BOOL retour = FALSE;

    // 0. on ajoute le message dans l'en-tete du protocole
    sprintf(protocole, "%s%s", protocole, message);

    // 1. on calcule le checksum de la trame
    checksum = calculerChecksum(protocole);

    // 2. on fabrique la trame
    sprintf(trame, "<ID01>%s%02X<E>", protocole, checksum);

    // 3. on transfere la trame

    // 3.1 on ouvre le port
    hPort = ouvrirPort(nomPort);
    if(hPort == INVALID_HANDLE_VALUE)
    {
        fprintf(stderr, "Erreur ouverture !\n");
        //return fd;
    }

    // 3.2 on envoie la trame
    retour = envoyer(hPort, &trame[0], strlen(trame));
    if(retour == FALSE)
    {
        fprintf(stderr, "Erreur transmission !\n");
        // ... ?
    }

    Sleep(DELAI);

    // 3.3 on receptionne l'acquittement
    retour = recevoir(hPort, reponse, LG_REPONSE);
    if(retour == FALSE)
    {
        fprintf(stderr, "Erreur reception !\n");
        // ... ?
    }
    else
    {
        printf("Reponse : %s\n", reponse);
    }

    // 3.4 on ferme le port
    fermerPort(hPort);

    // evite que la fenetre se ferme dans Dev-Cpp
    getch();

    return 0;
}
```

Vous pouvez compiler ce programme avec **GCC** (MinGW) ou à partir d'un EDI (Dev-C++, Code : :Blocks, ...).

On teste :

```
data packet :   0x3C 0x4C 0x31 0x3E 0x3C 0x50 0x41 0x3E 0x3C 0x46 0x45 0x3E 0x3C 0x4D 0x41 0x3E
                0x3C 0x57 0x43 0x3E 0x3C 0x46 0x45 0x3E 0x6D 0x65 0x73 0x73 0x61 0x67 0x65
checksum :  0x1F
-> envoyer (42/42) : <ID01><L1><PA><FE><MA><WC><FE>message1F<E>
<- recevoir (4/4) : 0x41 0x43 0x4B 0x4B
```



```
Reponse : ACKK
```

Dans le cas d'un mauvais checksum, on obtient :

```
-> envoyer (42/42) : <ID01><L1><PA><FE><MA><WC><FE>message00<E>
<- recevoir (4/4) : 0x4E 0x41 0x43 0x4B
Reponse : NACK
```

Code source : test-panneau-windows.c

Retour au sommaire

