Communiquer avec un périphérique branché sur un port série RS232 en C

1. Windows en utilisant l'API Win32

Aide: MSDN (MicroSoft Developer Network, http://msdn.microsoft.com/en-us/library/, rubrique MSDN Library\Windows Development\System Services\Device Services\Communications Resources).

a) Fonctions utiles

```
CreateFile()
GetCommState()
(GetCommTimeouts())
SetCommState()
(SetCommTimeouts())
(PurgeComm())
WriteFile()
ReadFile()
CloseHandle()
```

b) Description rapide des fonctions importantes

`CreateFile()`

Paramètres importants:

- Nom du port série (EX: \\.\COM4 mais attention, dans une fonction en C, le caractère `'\'` est un caractère spécial, il faut le précéder d'un autre `'\'` pour qu'il soit bien pris en compte, ce qui donnerait au final `"\\\\.\\COM4"`. De plus, on pourrait utiliser `"COM4"` tout simplement, mais pour les ports >= 10, il faut obligatoirement rajouter "\\.\") où est branché le périphérique avec lequel on veut communiquer. Les numéros des ports disponibles sur l'ordinateur sont visibles dans Panneau de configuration\Système\Matériel\Gestionnaire de périphériques\Ports (COM et LPT). Il reste alors à déterminer quel numéro est le bon s'il y en a plusieurs (on peut aussi modifier le numéro du port et d'autres options dans les propriétés avancées du port).

Valeur de retour: une sorte d'identifiant représentant le port ouvert, que l'on doit passer par la suite en paramètre de toutes les autres fonctions (`WriteFile()`, `CloseHandle()`,...)

Rôle: Ouvre le port série voulu et retourne un identifiant s'il n'y a pas de problèmes.

`GetCommState()`

Paramètres importants:

- Identifiant du port série (obtenu avec `CreateFile()`).
- Structure recevant les options actuelles du port série. Voir la documentation pour toutes les options.

Rôle: Obtient les options actuelles du port série (vitesse, nombre de bits de parité, nombre de bits de stop, taille d'une trame de données, contrôle de flux,...). Il est souvent pratique d'appeler cette fonction avant `SetCommState()` pour ne pas être obligé de se préoccuper aussi des options que l'on ne souhaite pas modifier.

`SetCommState()`

Paramètres importants:

- Identifiant du port série (obtenu avec `CreateFile()`).
- Structure contenant les options à appliquer au port série. Voir la documentation pour toutes les options.

Rôle: Modifie les options du port série (vitesse, nombre de bits de parité, nombre de bits de stop, taille d'une trame de données, contrôle de flux,...). Les options à modifier sont souvent la vitesse (baudrate), le nombre de bits de parité et de stop.

`WriteFile()`

Paramètres importants:

- Identifiant du port série (obtenu avec `CreateFile()`).
- Tableau d'entiers 8 bits (=1 octet ou byte) contenant les données à envoyer. Il faut consulter la documentation du périphérique pour savoir ce qu'il faut envoyer selon ce qu'on veut qu'il fasse et nous renvoie. Un protocole plus ou moins compliqué peut être utilisé. Note : les bits de parité et de stop liés au protocole RS232 ne sont pas à prendre en compte (le système saura les rajouter si on a spécifié les bonnes options avec `SetCommState()`), seules les données utiles sont à envoyer. Par contre, le périphérique peut fonctionner avec un protocole qui lui est propre (imbriqué dans le protocole RS232), mettant en jeu des notions de bits de parité, de stop, checksums,… qu'il faut alors considérer.
- Nombre d'octets à envoyer (nécessairement inférieur ou égal à la taille du tableau).
- Nombre d'octets réellement envoyés. Il faut vérifier cette valeur après l'appel de la fonction car toutes les données voulues ne sont pas toujours envoyées. Il faut alors rappeler la fonction pour qu'elle envoie les données manquantes (attention à ne pas renvoyer une 2ème fois les données déjà transmises).

Rôle: Envoie des données au périphérique via le port série. Attention au format des données (données constituées d'entiers entre -128 et 127, 0 et 255, sous forme de caractères, entiers hexadécimaux,...), aux temps de communication et au fait que le tableau donné en paramètre n'est pas forcément toujours envoyé entièrement en une fois. Ce comportement peut être modifié en réglant des options de timeouts appropriées. Voir la documentation pour plus d'informations.

`ReadFile()`

Paramètres importants:

- Identifiant du port série (obtenu avec `CreateFile()`).
- Tableau d'entiers 8 bits (=1 octet ou byte) recevant les données envoyées par le périphérique. Il faut consulter la documentation du périphérique pour savoir comment analyser les données reçues et y récupérer les valeurs qui nous intéressent. Un protocole plus ou moins compliqué peut être utilisé. Note : les bits de parité et de stop liés au protocole RS232 ne sont pas visibles (le système les a déjà utilisés et effacés si on a spécifié les bonnes options avec `SetCommState()`), seules les données utiles sont récupérées. Par contre, le périphérique peut fonctionner avec un protocole qui lui est propre (imbriqué dans le protocole RS232), mettant en jeu des notions de bits de parité, de stop, checksums,… qu'il faut alors considérer.
- Nombre d'octets à attendre (nécessairement inférieur ou égal à la taille du tableau).
- Nombre d'octets réellement reçus. Il faut vérifier cette valeur après l'appel de la fonction car toutes les données attendues ne sont pas toujours reçues. Il faut alors rappeler la fonction pour qu'elle essaye de récupérer les données manquantes (attention, rappeler `ReadFile()` ne demande pas au périphérique de renvoyer des données, on récupère juste des données qui étaient dans la file d'attente interne du port série).

Rôle: Reçoit des données venant du périphérique via le port série. Attention au format des données (données constituées d'entiers entre -128 et 127, 0 et 255, sous forme de caractères, entiers hexadécimaux...), aux temps de communication et au fait que les données attendues ne sont pas toujours reçues en 1 fois. Ce comportement peut être modifié en réglant des options de timeouts appropriées. Voir la documentation pour plus d'informations.

`CloseHandle()`

Paramètres importants:

- Identifiant du port série (obtenu avec `CreateFile()`).

Rôle: Ferme le port série.

---

c) Fonctions potentiellement utiles

`GetCommTimeouts()`

Paramètres importants:

- Identifiant du port série (obtenu avec `CreateFile()`).
- Structure recevant les options actuelles de limite de temps d'attente d'une donnée du port série. Voir la documentation pour plus d'informations.

Rôle: Obtient les options actuelles de limite de temps d'attente d'une donnée du port série. Il est souvent pratique d'appeler cette fonction avant `SetCommTimeouts()` pour ne pas être obligé de se préoccuper aussi des options que l'on ne souhaite pas modifier.

`SetCommTimeouts()`

Paramètres importants:

- Identifiant du port série (obtenu avec `CreateFile()`).
- Structure contenant les options de limite de temps d'attente d'une donnée à appliquer au port série. Voir la documentation pour plus d'informations.

Rôle: Modifie les options actuelles de limite de temps d'attente d'une donnée du port série.

`PurgeComm()`

Paramètres importants:

- Identifiant du port série (obtenu avec `CreateFile()`).

Rôle: Supprime les données non envoyées ou non reçues qui sont dans la file d'attente interne du port série.

---

d) Exemple

Supposons qu'on ait un périphérique que l'on veut brancher sur un ordinateur portable. Cet ordinateur n'ayant pas de port série RS232 accessible (connecteur au format DB9), on utilise un convertisseur USB-RS232. Après avoir installé les drivers de ce convertisseur et l'avoir branché sur un port USB, on note le numéro du port COM qui apparaît dans Panneau de configuration\Système\Matériel\Gestionnaire de périphériques\Ports (COM et LPT) (EX: COM4). Ce périphérique envoie la chaîne de caractères `"Hello"` si on lui envoie l'entier 63 (code ASCII de `'?'`). Il fonctionne à une vitesse de 9600 bps, envoie des trames avec 8 bits de données (1 octet), pas de bit de parité et 1 bit d'arrêt (cette configuration est la plus classique et est parfois appelée 8N1 (8 pour 8 data bits, N pour No parity, 1 pour 1 stop bit)). Voir Win32APIRS232Device_vs2008.zip et Win32APIRS232Device_Qt.zip.

2. Linux en utilisant l'API POSIX

Aide: Linux Man Pages (commande man, http://pwet.fr/man/linux rubrique Fonctions des bibliothèques\posix), voir aussi e.g. https://www.cmrr.umn.edu/~strupp/serial.html, http://digilander.libero.it/robang/rubrica/serial.htm.

a) Fonctions utiles

```
open()
(fcntl())
tcgetattr()
cfmakeraw()
cfsetospeed()
cfsetispeed()
tcsetattr()
(tcflush())
```

```
write()
read()
close()
```

Sous Linux, les ports série RS232 sont souvent nommés /dev/ttyS0, /dev/ttyS1… ou encore /dev/ttyUSB0 si on passe par un convertisseur USB-RS232, ainsi que parfois /dev/ttyACM0. Il peut aussi être intéressant d'utiliser des commandes telles que stty pour configurer un port série pour une application qui ne gérerait pas bien les paramètres par défaut (e.g. utiliser la commande stty -F /dev/ttyUSB0 -a pour connaitre les options par défaut du port /dev/ttyUSB0).

3. Exemples de code fonctionnant sous Windows et Linux

RS232PortVirtualPairTest.zip: RS232PortTest1 et RS232PortTest2 sont 2 programmes communiquant entre eux via un port série RS232 (envoi/réception d'un buffer). Pour les tester rapidement, on peut par exemple utiliser Virtual Serial Port Emulator (ou encore com0com) sous Windows ou socat (en tapant une commande du type socat -d -d pty,raw,echo=0 pty,raw,echo=0 et en regardant le nom des 2 ports attribués, par exemple /dev/pts/1 et /dev/pts/2) sous Linux pour créer une paire de ports série RS232 virtuels reliés entre eux. Il est aussi possible de faire le test entre 2 PC différents reliés par un câble série RS232 croisé (reliant la pin TX de l'un sur le RX de l'autre et inversement). Il faut noter aussi que si on veut tester une liaison série RS232 entre autre chose que des PC (par exemple une carte avec processeur ARM sous Linux…), il faut bien vérifier les niveaux de tensions des signaux RS232 avant de faire un branchement car il existe plusieurs variantes: ±12V (sur le port DB9 de la plupart des PC ou convertisseur USB-RS232 classiques), ±5V, ±3.3V…

Voir aussi https://github.com/ENSTABretagneRobotics/Hardware-CPP .