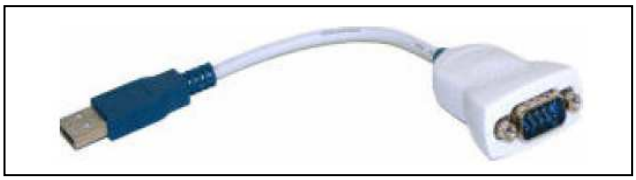
TP

La liaison RS232

Nos PCs ne possèdent pas de port série ; on utilise un adaptateur USB/RS232 construit autour du composant FT232RL de chez FTDI qui émule une liaison série RS232 :

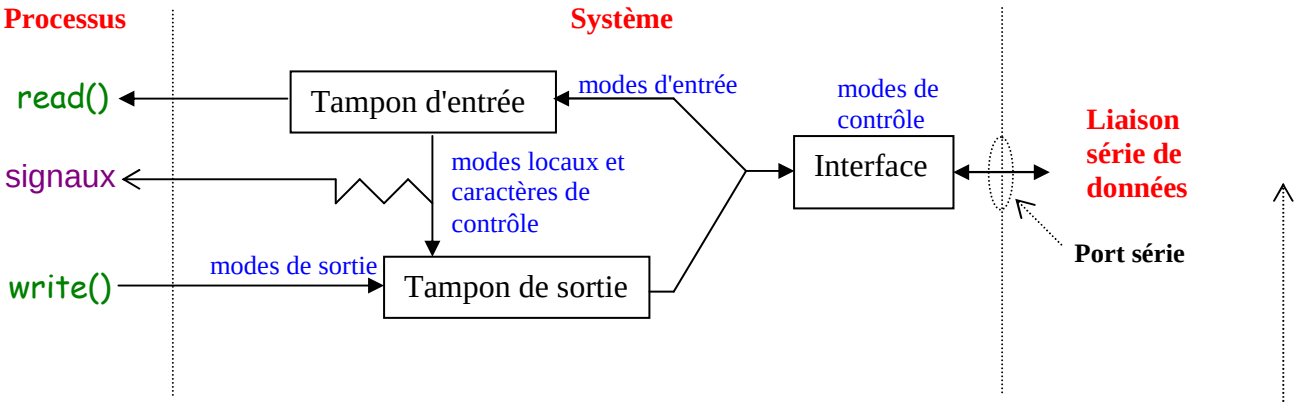


Sous Linux, chaque port série est associé à un fichier situé dans le répertoire `/dev`. Le répertoire /dev est la racine de la sous-arborescence des fichiers spéciaux associés aux ressources du système.
Ainsi, quand on connecte un adaptateur USB/RS232 à un PC, **le port série virtuel obtenu est associé au fichier spécial /dev/ttyUSB0**. Si on connecte un second adaptateur USB/RS232 au même PC, on obtient un 2ème port série virtuel associé au fichier spécial /dev/ttyUSB**1**, etc...

Le paramétrage d'une liaison série

Sous Linux, la configuration d'une voie de communication entre le système et une extrémité d'une liaison série de données (matérialisée par un port série) est réalisée de la même façon que la configuration d'une voie de communication entre le système et un terminal (historiquement, un terminal correspondait à un terminal physique connecté à un port de communication série; de nos jours, un terminal correspond à une application jouant le rôle de terminal physique).

Le schéma suivant trace les grandes lignes de la gestion d'une **voie de communication entre le système et une extrémité d'une liaison série de données, matérialisée par un port série.**



Dans la suite de ce paragraphe, on considère qu'un terminal (écran + clavier) est physiquement connecté à l'autre extrémité de la liaison série de données.

Les modes de contrôle correspondent à des informations de contrôle de la transmission relatives au niveau **matériel**. Ils permettent notamment de **configurer la vitesse de transmission** (exprimée en baud), et **la structure de la trame** (nombre de bits de donnée, parité, nombre de bits de stop).

Les modes d'entrée définissent un certain nombre de traitements à appliquer aux caractères en provenance du terminal (reçus via le signal Rx dans le cas d'une liaison RS232), qui sont ensuite accessibles par les processus par l'intermédiaire de la fonction **read().**

Inversement, les caractères écrits à l'aide de la fonction **write()** par les processus à destination d'un terminal subissent les transformations spécifiées par les modes de sortie avant leur émission (via le signal Tx dans le cas d'une liaison RS232).

Les modes locaux définissent en particulier le comportement général d'une voie de communication en ce qui concerne les caractères de contrôle et **déterminent le fonctionnement des entrées sorties** :

- **ISIG** : dans ce mode, certains caractères sont considérés comme des caractères de contrôle : leur réception provoque l'envoi de signaux à certains processus. Par exemple, dans ce mode, la frappe du caractère <intr> (CTRL C par défaut) entraîne l'envoi du signal SIGINT à tous les processus en avant plan du terminal, ce qui provoque par défaut leur terminaison.
- **ECHO** : dans ce mode, tous les caractères en provenance du terminal sont, après les transformations définies par les modes d'entrée, insérés dans le flux de sortie à destination du terminal. L'absence de cet indicateur fait que ce qui est tapé au clavier n'est normalement pas visible sur l'écran. C'est ainsi par exemple que les mots de passe sont introduits.
- **ICANON** : il correspond au **mode canonique** de fonctionnement (celui d'un terminal utilisé par un utilisateur en mode intéractif). Dans ce mode, le **tampon de lecture**, dans lequel les caractères en provenance du terminal sont stockés, est **structuré en lignes**, une ligne étant une suite de caractères terminée par le caractère ASCII <LF> (Line Feed) ou <NL> (New Line). Cela signifie que **les caractères lus au cours d'une opération de lecture sur le terminal** (appel de la fonction **read()**) **sont extraits dans une ligne et une seule**. Donc, tout caractère qui n'est pas suivi d'un caractère de fin de ligne ne peut être accessible en lecture et une opération de lecture ne peut lire "à cheval" sur plusieurs lignes.

Lorsque l'indicateur ICANON est absent, la gestion de la voie de communication est assurée dans le **mode non canonique**. Dans ce cas, la structure de ligne ne définit plus le critère d'accessibilité des caractères. Les **critères d'accessibilité en lecture** des caractères en provenance du terminal sont alors **fixés par 2 paramètres de configuration nommés MIN et TIME** selon 4 cas :

- si MIN > 0 et TIME > 0, les caractères tapés au terminal (reçus via Rx dans le cas d'une liaison RS232) sont disponibles à la lecture soit lorsque MIN caractères ont été reçus, soit lorsqu'un intervalle de temps de TIME*0.1 secondes s'est écoulé depuis la réception du dernier caractère. Dans cette configuration, au moins un caractère doit être reçu pour débloquer un appel de la fonction **read().**
- si MIN > 0 et TIME = 0, les caractères tapés au terminal (reçus via Rx dans le cas d'une liaison RS232) sont disponibles à la lecture lorsque MIN caractères ont été reçus.
- si MIN = 0 et TIME > 0, la temporisation (TIME*0.1 secondes) s'applique à l'appel de la fonction **read()** et la lecture est satisfaite dès qu'un caractère est reçu. Si le délai expire avant qu'un caractère ne soit reçu, l'appel de la fonction **read()** se termine sans qu'un caractère soit lu.
- si MIN = 0 et TIME = 0, un appel de la fonction **read()** est non bloquant : soit il y avait quelque chose à lire et la lecture se produit normalement, soit il n'y avait rien à lire et l'appel de la fonction **read()** ne lit pas de caractère.

Remarque : les valeurs les plus couramment utilisés en mode non canonique pour les paramètres MIN et TIME sont respectivement **1** et **0**. Il s'agit d'un mode ou **chaque caractère reçu est immédiatement accessible en lecture**.

Documents à rendre :

Vous devez rendre :
- **un document LibreOffice Writter** avec les réponses aux questions ;
- et vos programmes sources.

Travail à réaliser :

Partie I : Utilisation de la commande stty pour configurer la voie de communication avec le terminal associé à l'entrée standard

La commande stty

Synopsis :

stty [-F DEVICE] [-a]
stty [-F DEVICE] [SETTING] ...

- les [] signifie que l'argument correspondant est optionnel.
- les ... indique qu'il peut y avoir plusieurs settings.

L'option **-a** permet de **visualiser** la valeur de tous les paramètres de la voie de communication :
- ✓ entre le système et le terminal associé à l'entrée standard par défaut,
- ✓ ou entre le système et le device précisé à l'aide de l'option -F.

Un mode précédé d'un - est désactivé; il est activé dans le cas contraire.

La seconde forme de la commande permet de **modifier** les paramètres de la voie de communication :
- ✓ entre le système et le terminal associé à l'entrée standard par défaut,
- ✓ ou entre le système et le device précisé à l'aide de l'option -F.

Les modifications à apporter sont indiqués à l'aide du ou des settings.
Les settings peuvent être interprétés de différentes manières :
- comme la demande d'activation d'un mode de fonctionnement particulier : l'argument est alors le nom symbolique du mode de fonctionnement;
- comme la demande de désactivation d'un mode de fonctionnement particulier : l'argument est alors constitué du caractère - suivi du nom symbolique du mode de fonctionnement.

Si la commande stty est utilisée sans arguments, seules les valeurs d'un certain nombre de paramètres sont visualisées.

Pour plus d'informations, consulter le manuel de la commande stty (**man stty**).

Remarque : suite à un paramétrage "hasardeux", le terminal peut se trouver dans un état très "étrange" où rien ne semble fonctionner normalement. La commande "stty sane" permet de rétablir les différents paramètres de la voie de communication entre le système et le terminal à des valeurs raisonnables permettant une interaction "normale" avec le système.

Q1) Dans un fenêtre terminal, taper la commande "stty -a" pour visualiser tous les paramètres de la voie de communication entre le système et le terminal associé à l'entrée standard. Les modes ISIG, ICANON et ECHO sont-ils activés?

Q2) Etude du mode ECHO

Donner et exécuter la commande qui permet de désactiver le mode ECHO de la voie de communication entre le système et le terminal associé à l'entrée standard.
Taper ensuite la commande "ls". Conclusion? Pour finir, réactiver le mode ECHO.

Q3) Etude du mode ISIG

Réaliser un programme qui contient une boucle infinie. Lancer son exécution dans une fenêtre terminal. Pour stopper le processus exécutant ce programme, taper le caractère <intr> (**CTRL C** par défaut) : cela entraîne l'envoi du signal SIGINT au processus ce qui provoque sa terminaison.

Donner et exécuter la commande qui permet de désactiver le mode ISIG de la voie de communication entre le système et le terminal associé à l'entrée standard.

Relancer l'exécution du programme contenant la boucle infinie. Essayez de terminer à nouveau son exécution en tapant le caractère <intr> (**CTRL C**). Conclusion ?

Q4) Etude du mode ICANON

Soit le programme source suivant :

```
#include <stdio.h>

int main()
{
        char c;
        c = fgetc(stdin);
        if (c != EOF)
        {
                printf("Code ASCII en hexa du caractère lu : %x, caractère lu : %c\n", c, c);
        }
        else
        {
                printf("EOF\n");
        }
        return 0;
}
```

Générer l'exécutable correspondant en utilisant la commande gcc dans une fenêtre terminal.

a) Dans la fenêtre terminal (en mode canonique), lancer l'exécutable. Dans ce mode, que faut-il faire pour rendre un caractère accessible en lecture?

b) Donner et exécuter la commande qui permet de désactiver le mode ICANON de la voie de communication entre le système et le terminal associé à l'entrée standard. Par défaut, quelles sont les valeurs des paramètres MIN et TIME? Exécuter le programme. Conclusion?

c) Donner et exécuter la commande qui permet d'obtenir MIN = 4 et TIME = 0. Exécuter le programme. A quelle condition, les caractères tapés au terminal sont-ils accessibles (ou disponibles) en lecture?

d) Donner et exécuter la commande qui permet d'obtenir MIN = 4 et TIME = 50. Exécuter le programme. A quelles conditions, les caractères tapés au terminal sont-ils accessibles (ou disponibles) en lecture?

e) Donner et exécuter la commande qui permet d'obtenir MIN = 0 et TIME = 50. Exécuter le programme. A quelle condition, la fonction fgetc() renvoie-t-elle un caractère lu? Que se passe-t-il dans le cas contraire?

f) Donner et exécuter la commande qui permet d'obtenir MIN = 0 et TIME = 0. Exécuter le programme. L'utilisateur a t-il le temps de saisir un caractère? Que renvoie la fonction fgetc()? Modifier le programme pour que l'utilisateur ait le temps de saisir un ou plusieurs caractères (utiliser la fonction sleep() qui nécessite l'inclusion du fichier <unistd.h>).

g) Synthèse :

En cas d'échec, que renvoie la fonction fgetc()?
Compléter le tableau suivant.

	Lecture bloquante ou non bloquante ?	Un appel à fgetc() peut-il échouer ? Si oui, à quelle condition ?	Critère(s) d'accessibilité des caractères reçus ?
Mode canonique			
Mode non canonique MIN = 1, TIME = 0			
Mode non canonique MIN > 1, TIME = 0			
Mode non canonique MIN > 1, TIME > 0			
Mode non canonique MIN = 0, TIME > 0			
Mode non canonique MIN = 0, TIME = 0			

Partie II : Utilisation des commandes stty, echo et cat

Dans cette partie, vous allez utiliser les commandes :
- **stty** pour configurer une liaison série RS232 (ou plus exactement pour configurer la voie de communication entre le système et une extrémité de la liaison série RS232),
- **echo** pour transmettre des caractères via un port série,
- **cat** pour lire les données reçues via un port série

On souhaite ici échanger des caractères entre un PC émetteur et un PC récepteur via une liaison RS232, ce qui implique que vous devez travailler en binôme.

Q1) Connexions

Connecter un adaptateur USB/RS232 sur votre PC. Quel type de câble faut-il utiliser pour relier les 2 prises SUBD mâle à 9 contacts ?

Rappel : quand on connecte un adaptateur USB/RS232 à un PC, **le port série virtuel obtenu est associé au fichier spécial /dev/ttyUSB0**.

Q2) Configuration de la liaison série RS232 :

Configurer la voie de communication entre le système et le port série virtuel en mode non canonique sans echo. Désactiver les modes ISIG et ECHOE.
Fixer les valeurs des paramètres MIN et TIME respectivement à 1 et à 0.

Activer les modes HUPCL, CLOCAL.

Donner et exécuter la commande qui permet de fixer la vitesse de transmission à 19200 baud avec une structure de trame 8E1.
Dans le cas on l'on souhaite recevoir des caractères, activer le mode CREAD.

Activer le mode IGNBRK. Afin de rejeter les caractères contenant une erreur de parité, activer les modes IGNPAR, INPCK et PARMRK.

Q3) Echange de caractères

Sur le PC récepteur, exécuter la commande "cat /dev/ttyUSB0" afin de **lire** le fichier spécial associé au port série virtuel.

Sur le PC émetteur, exécuter la commande "echo *message* > /dev/ttyUSB0" afin d'**écrire** le *message* dans le fichier spécial associé au port série virtuel. Vous pouvez exécuter cette commande plusieurs fois pour envoyer différents messages.
Remarque : la commande echo permet écrire un message sur la sortie standard (par défaut la sortie standard est connectée au terminal où la commande est exécutée). La directive "> /dev/ttyUSB0" permet de rediriger le sortie standard du processus vers le fichier /dev/ttyUSB0.


Fixer une parité impaire d'un "seul coté". Essayer d'échanger des caractères entre les 2 PCs. Conclusion?


Fixer à nouveau une parité paire pour les 2 voies de communication et modifier la vitesse de transmission d'un "seul coté". Essayer d'échanger des caractères entre les 2 PCs. Conclusion?

Partie III : Utilisation de minicom

Minicom est un programme de contrôle de modem et d'émulation de terminal pour les Unix-like. Il peut être comparé à HyperTerminal dans Windows.
Il est souvent utilisé pour la configuration d'équipements réseaux (switch, routeur) via leur port console.

On souhaite ici échanger des caractères entre 2 PCs à la fois émetteur et récepteur, via une liaison RS232 : des 2 cotés, on souhaite pouvoir émettre et recevoir des données. Vous devez travailler en binôme.

Q1) La commande "minicom -D /dev/ttyUSB0 -s" permet :
- avec l'option -D de préciser le port série à utiliser;
- avec l'option -s d'entrer en mode de configuration.

Lancer cette commande.

Q2) Des 2 cotés, configurer la voie de communication entre le système et le port série :
- vitesse de transmission : 19200 Bd
- structure de trame 8O2.

Q3) Finaliser la configuration de minicom pour que les messages envoyés et reçus s'affichent correctement.
Remarque : vous pouvez appuyer sur **CTRL-A + Z** pour afficher l'aide.
Appeler le professeur pour vérification.

Partie IV : Programmation

I. La norme POSIX

Les quatre premières lettres forment l'acronyme de **P**ortable **O**perating **S**ystem **I**nterface, et le X exprime l'héritage UNIX de l'Interface de programmation.
Cette norme propose la **spécification d'un ensemble de fonctions** permettant de **solliciter ou d'accéder aux services de base d'un système d'exploitation** (tels que les opérations d'entrées-sorties et les opérations relatives aux processus et à leur communication). L'objectif est de permettre le développement d'applications portables au niveau du code source.

Dans les systèmes Unix, l'interface POSIX est une interface directe avec le système (on la qualifie parfois de **native**) : les fonctions qu'elle définit sont presque toutes des **appels système**. De manière générale, les appels système permettent de réaliser, sous le contrôle du noyau du système correspondant, des opérations "dangereuses" dans le sens qu'elles nécessitent la consultation et la modification des données du noyau (variables, tables).

La bibliothèque standard du langage C contient elle aussi des fonctions d'entrées-sorties (fopen(), fread(), fwrite(), fclose(), ...). Dans les systèmes Unix, cette bibliothèque constitue **une couche au dessus** des appels système correspondant aux fonctions POSIX qui sont décrites dans la suite du TP. Les fonctions de la bibliothèque standard du langage C masquent donc les spécificités du système et permettent d'obtenir un niveau de portabilité plus élevé.

II. Le modèle de structure termios

Pour la norme POSIX, l'ensemble des paramètres d'une voie de communication entre le système et un terminal (ou entre le système et un port série) est rassemblé dans le modèle de structure *termios*, déclarée dans le fichier d'entête standard <termios.h> (qu'il faudra inclure).

La manipulation de variables du type "modèle de structure *termios*" permettra de faire avec un programme écrit en C, ou en C++, ce que l'on peut faire avec la commande stty.

Le modèle de structure *termios* est déclaré de la façon suivante :

```
struct termios
{
        tcflag_t  c_iflag;      // input modes
        tcflag_t  c_oflag;      // output modes
        tcflag_t  c_cflag;      // control modes
        tcflag_t  c_lflag;      // local modes
        cc_t      c_cc[NCCS];   // control chars
};
```

Chacun des 4 premiers champs (de type tcflag_t) est constitué d'une suite de bits; le positionnement de chacun de ces bits correspond à un mode ou à un traitement particulier.

Un symbole correspondant à un **entier non signé avec un seul bit à 1** est macrodéfini pour chaque mode. Pour activer un mode, il suffit donc de mettre à 1 le bit correspondant. De façon similaire, pour désactiver un mode, il suffit de mettre à 0 le bit correspondant.

Le dernier champ est un tableau qui contient les valeurs des caractères de contrôle.

Le fichier /usr/include/bits/termios.h contient notamment les déclarations des différents types, et les macrodéfinitions des symboles. Visualiser son contenu avec gedit. Comme les valeurs numériques des constantes entières sont précédées d'un 0, elles sont **écrites en octal.**

Remarque : en langage C ou C++ :
- par défaut, les constantes numériques sont écrites en décimal;
- quand elles sont préfixées par 0x, elles sont écrites en hexadécimal;
- quand elles sont préfixées par 0, elles sont écrites en octal.

Exemple :	0x31 =	% 0011 0001 =	49 =	061
	Hexa	Binaire naturel	Décimal	Octal

Q1) Si le champ c_iflag vaut 3093 en décimal, déterminer les modes d'entrée qui sont actifs.

III. Les opérateurs logiques bit à bit : &, | , ^ , ~

A priori, **ces opérateurs ne peuvent porter que sur des types entiers**. Toutefois, compte tenu des règles de conversions non dégradantes (char → short int → int), ils pourront également s'appliquer à des caractères (mais le résultat sera un entier).
La liste de ces opérateurs se compose de trois opérateurs binaires (manipulant 2 opérandes) :

- & : ET (bit à bit);
- | : OU inclusif (bit à bit);
- ^ : OU exclusif (bit à bit).

et d'un opérateur unaire (manipulant un seul opérande) :

- ~ : Complément à 1 (bit à bit).

Dans la table de vérité de gauche, a et b correspondent à des bits de même rang des 2 opérandes (i.e. des 2 entiers). L'opérateur ~ ne manipule qu'un seul opérande (un seul entier) :

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

a	~a
0	1
1	0

Exemple :
Soit n et p 2 variables de type **short int** : chaque variable est stockée sur 16 bits et utilise un code complément à 2 sur 16 bits pour coder un entier signé.

Soit	n = 1390	= 0x056E	= 0b0000 0101 0110 1110
et	p = 946	= 0x03B2	= 0b0000 0011 1011 0010
=>	n & p = 290	= 0x0122	= 0b0000 0001 0010 0010
	n | p = 2046	= 0x07FE	= 0b0000 0111 1111 1110
	n ^ p = 1756	= 0x06DC	= 0b0000 0110 1101 1100
	~n = -1391	= 0xFA91	= 0b1111 1010 1001 0001
	~p = -947	= 0xFC4D	= 0b1111 1100 0100 1101

Q2) Soit n et p 2 variables de type **short int**, avec n = 747 et p = 32605.
Donner les valeurs en décimal, hexadécimal et binaire des expressions :

- n | p
- n ^ p
- n & p
- ~n

Q3) Soit une variable *term* de type modèle de structure *termios*.
Ecrire une instruction qui permet d'activer le mode ICANON sans modifier les autres modes locaux de la variable.
Ecrire une instruction qui permet de désactiver le mode ISIG sans modifier les autres modes locaux de la variable.

IV. Les fonctions à utiliser

Remarque : toutes les fonctions présentées ci-dessous renvoient -1 en cas d'échec.

IV.1. L'ouverture d'un fichier : la fonction open()

Fichiers à inclure et prototype de cette fonction :

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char * reference, int mode, ... /* mode_t droits */ );
```

Cette fonction permet d'ouvrir le fichier de ***reference*** donnée pour les opérations spécifiées par le ***mode*** d'ouverture (le paramètre supplémentaire correspondant aux *droits* d'accès n'est utile qu'en cas de création de fichier).

En cas de succès, un appel à cette fonction renvoie **un entier supérieur ou égal à 0** et cet entier correspond à un **descripteur qui est associé au fichier ouvert** et qui, ensuite, **devra être utilisé pour réaliser les opérations souhaitées sur ce fichier.**
En cas d'échec, la fonction *open()* renvoie -1.

Le *mode* d'ouverture doit contenir **un seul** des 3 symboles macrodéfinis suivantes :

Symbole	Mode d'ouverture
O_RDONLY	En lecture seule
O_WRONLY	En écriture seule
O_RDWR	En lecture et écriture

Exemple :

```
int fd;             /* pour file descriptor */

fd = open("/dev/ttyUSB0",O_RDONLY);        /* ouverture en lecture seule : récepteur */
if (fd == -1)                              /* Ne pas oublier de tester si l'ouverture a réussi */
{
        printf ("Erreur d'ouverture!\n");
}
```

IV.2. La configuration d'une voie de communication entre le système et un terminal (ou un port série)

IV.2.1 La fonction :

```
#include <termios.h>
int tcgetattr( int descripteur, struct termios *pTermios);
```

extrait les paramètres courants appliqués à la voie de communication correspondant au ***descripteur*** et les copie en mémoire à l'adresse ***pTermios*.** Il s'agit donc ici d'un accès en consultation (get).

IV.2.2 La fonction :

```
#include <termios.h>
int tcsetattr( int descripteur, int option, struct termios *pTermios);
```

installe les paramètres contenus en mémoire à l'adresse ***pTermios*** pour la voie de communication entre le système et le terminal (ou le port série) associé au ***descripteur*** spécifié.
Il s'agit donc ici d'un accès en modification (set).
Le paramètre ***option*** peut prendre l'une des valeurs :

Symbole	Effet
TCSANOW	Les modifications opèrent immédiatement
TCSADRAIN	Les modifications ne seront effectivement qu'après vidage du tampon de sortie
TCSAFLUSH	Les modifications ne seront effectivement qu'une fois le vidage du tampon de sortie effectué, de plus les caractères reçus, mais non encore lus, sont oubliés.

Une pratique usuelle consiste à :
- dans un premier temps, extraire les valeurs courantes des paramètres d'une voie de communication,
- préparer en mémoire un nouvel état en modifiant les champs en fonction du résultat souhaité,
- puis, installer ce nouvel état.

Remarque : les modifications apportées aux paramètres d'une voie de communication ont un effet sur toutes les opérations réalisées sur cette voie via n'importe quel descripteur de n'importe quel processus.

IV.2.3 Les 2 fonctions :

```
#include <termios.h>
int cfsetospeed(struct termios *pTermios, speed_t vitesse);     /* vitesse en sortie */
int cfsetispeed(struct termios *pTermios, speed_t vitesse);     /* vitesse en entrée */
```

permettent d'encoder la ***vitesse*** en entrée et en sortie dans la variable (de type *struct termios*) située à l'adresse ***pTermios***.

Remarques :
- un appel à l'une de ces fonctions ne modifie pas effectivement la vitesse : la modification des vitesses n'aura lieu qu'à l'installation des nouveaux paramètres contenus dans la variable de type *struct termios*, c'est à dire suite à un appel de la fonction **tcsetattr();**
- pour les vitesses de transmission, il faut utiliser un symbole macrodéfini dans le fichier /usr/include/bits/termios.h (**le consulter**). Exemple : B9600

IV.2.4 Modification d'un élément du champ c_cc d'une variable de type *struct termios*

Ce champ est un tableau qui contient :
- les valeurs des caractères de contrôle;
- les valeurs de certains paramètres comme MIN et TIME

Chacun des éléments du tableau c_cc possède un nom symbolique (MIN par exemple) et sa position dans le tableau (c'est à dire son indice) est symboliquement désignée par son nom (en majuscules) précédé du caractère V (ainsi c_cc [VMIN] contient la valeur du paramètre MIN).

IV.2.5 Exemple

```
int fd, res;
struct termios termiosI, termiosM;

fd = open("/dev/ttyUSB0",O_RDONLY);          /* ouverture en lecture seule : récepteur */
if (fd == -1)                                /* Ne pas oublier de tester si l'ouverture a réussi */
{
        printf ("Erreur d'ouverture!\n");
}

res = tcgetattr(fd, &termiosI);              /* Récupération dans la variable termiosI des paramètres
                                                de la voie de communication entre le système et le port
                                                série associé au descripteur fd */

if (res == -1)              /* Ne pas oublier de tester si l'appel de la fonction a réussi*/
{
        printf ("Pb d'accès en consultation aux paramètres de la voie de communication!\n");
}

termiosM = termiosI;        /* la variable termiosI ne sera pas modifiée afin de conserver la
                               configuration initiale de la voie de communication. Ainsi, on
                               pourra ainsi la réinstaller plus tard si on le désire */
```

```
/* parmi les modes locaux, désactivation du mode canonique sans modifier les autres modes */
termiosM.c_lflag &= ~ICANON;

/* parmi les modes d'entrée, activation du mode IGNBRK sans modifier les autres modes */
termiosM.c_iflag |= IGNBRK;

/* paramètres MIN et TIME fixés respectivement à 4 et 0 */
termiosM.c_cc[VMIN] = 4;
termiosM.c_cc[VTIME] = 0;

/* vitesse de la transmission fixée à 9600 Bd */
res = cfsetospeed(&termiosM, B9600);
if (res == -1)
{
        printf ("Pb lors de la modification de la vitesse de sortie!\n");
}
res = cfsetispeed(&termiosM, B9600);
if (res == -1)
{
        printf ("Pb lors de la modification de la vitesse d'entrée!\n");
}

/* Installation des nouveaux paramètres de la voie de communication entre le système et le port série */
res = tcsetattr(fd, TCSANOW, &termiosM) ;
if (res == -1)              /* Ne pas oublier de tester si l'appel de la fonction a réussi*/
{
    printf ("Pb lors de l'installation des nouveaux paramètres de la voie de communication!\n");
}

/* Utilisation du port série */
...

/* A la fin du programme, réinstallation des paramètres initiaux de la voie de communication entre le système et le port série */

res = tcsetattr(fd, TCSANOW, &termiosI) ;
if (res == -1)              /* Ne pas oublier de tester si l'appel de la fonction a réussi*/
{
        printf ("Pb lors de la réinstallation des paramètres initiaux!\n");
}
```

Remarque : la réinstallation des paramètres initiaux de la voie de communication est :
- NON OBLIGATOIRE pour un port série,
- FORTEMENT CONSEILLÉE pour un terminal.

IV.3. L'écriture de caractères dans un fichier

La fonction write() :

```
#include <unistd.h>
ssize_t write(int descripteur, void * tampon, size_t nombre);
```

Un appel de cette fonction correspond à une demande d'écriture, dans le fichier associé au ***descripteur***, du ***nombre*** de caractères lus à partir de l'adresse ***tampon*** dans l'espace d'adressage du processus.

Remarques :
- le type size_t correspond à un entier non signé;
- le type ssize_t correspond à un entier signé.

Dans le cas d'un fichier de données sur le disque (fichier régulier), l'algorithme correspondant est le suivant :

- La fonction renvoie -1 s'il y a une erreur de paramètre (descripteur inconnu ou ne correspondant pas à une ouverture en écriture).
- S'il n'y a pas d'erreur de paramètre, il y a écriture dans le fichier à partir de la position courante (offset) des ***nombre*** caractères lus à l'adresse ***tampon*** dans l'espace d'adressage du processus. Le nombre de caractères écrits est renvoyé : une valeur inférieure à ***nombre*** indique une erreur. En cas d'écriture, la position courante (offset) associée au ***descripteur*** est augmentée du nombre de caractères écrits.

Exemple :

```
char message [30] = "Bonjour";
int nbecrits;

nbecrits = write(fd, message, 7);        /* le caractère nul n'est pas écrit */
if (nbecrits == -1)
{
        printf("Erreur lors de l'écriture!\n");
}
else
{
        printf("Nombre de caractères écrits : %d\n", nbecrits);
}
```

IV.4. La lecture de caractères dans un fichier

La fonction read() :

```
#include <unistd.h>
ssize_t read(int descripteur, void * tampon, size_t nombre);
```

Un appel de cette fonction correspond à la demande de lecture d'au plus ***nombre*** caractères via le ***descripteur***. Ce descripteur doit être associé à un fichier ouvert (suite à un appel de la fonction *open*()) dans un mode autorisant la lecture (O_RDONLY ou O_RDWR).
Les caractères, lus dans le fichier, sont écrits dans l'ordre à partir de l'adresse spécifiée par ***tampon***. Cette adresse est supposée correspondre à un espace de taille suffisante permettant au processus l'écriture du ***nombre*** de caractères spécifié.

Dans le **cas d'un fichier de données sur le disque** (fichier régulier), l'algorithme correspondant est le suivant :

- La fonction renvoie -1 s'il y a une erreur de paramètre (descripteur inconnu ou ne correspondant pas à une ouverture en lecture).
- S'il n'y a pas d'erreur de paramètre :
  - ✓ si l'offset (position courante) est inférieur à la taille du fichier, la fonction lit des caractères dans le fichier à partir de la position courante, jusqu'à ce que le ***nombre*** de caractères demandés aient été lus ou que la fin de fichier soit atteinte. La fonction renvoie alors le nombre de caractères effectivement lus et la valeur de l'offset est augmentée de ce nombre.
  - ✓ si l'offset est supérieur ou égal à la taille du fichier, la fonction renvoie 0.

S'il n'y a pas d'erreur de paramètre, dans le **cas d'un fichier associé à un terminal ou à un port série**, tout dépend du paramétrage de la voie de communication :

➢ En mode **canonique**:

- ✓ si des caractères reçus sont accessibles (rappel : un caractère reçu pour être accessible doit être placé dans une **ligne**, c'est à dire suivi d'un caractère de fin de ligne), la fonction lit au maximum ***nombre*** caractères dans la première ligne reçue. La fonction renvoie le nombre de caractères lus.
  Remarques : - un appel à la fonction read() ne peut pas lire à cheval sur plusieurs lignes;
  - le caractère de fin de ligne (LF) est accessible;
  - la lecture d'une ligne est terminée quand son caractère de fin de ligne a été lu.
- ✓ s'il n'y a pas de caractère accessible, l'appel est **bloquant** et la fonction attend que des caractères reçus deviennent accessibles, c'est à dire au moins la réception d'un caractère de fin de ligne. A ce moment, la nouvelle ligne reçue est analysée, la fonction y lit au maximum ***nombre*** caractères et elle renvoie le nombre de caractères lus.

➢ En mode **non canonique**:

- ✓ si MIN > 0 et TIME > 0, les caractères tapés au terminal (reçus via Rx dans le cas d'une liaison RS232) sont disponibles à la lecture soit lorsque MIN caractères ont été reçus, soit lorsqu'un intervalle de temps de TIME*0.1 secondes s'est écoulé depuis la réception du dernier caractère. Dans cette configuration, au moins un caractère doit être reçu pour débloquer un appel de la fonction read().
- ✓ si MIN > 0 et TIME = 0, les caractères tapés au terminal (reçus via Rx dans le cas d'une liaison RS232) sont disponibles à la lecture lorsque MIN caractères ont été reçus. Dans cette configuration, MIN caractères doivent être reçus pour débloquer un appel de la fonction read().
- ✓ si MIN = 0 et TIME > 0, la temporisation (TIME*0.1 secondes) s'applique à l'appel de la fonction read() et la lecture est satisfaite dès qu'un caractère est reçu. Si le délai expire avant qu'un caractère ne soit reçu, l'appel de la fonction read() se termine et renvoie la valeur 0 (ce qui signifie 0 caractère lu).
- ✓ si MIN = 0 et TIME = 0, un appel de la fonction read() est non bloquant : soit il y avait quelque chose à lire et la lecture se produit normalement, soit il n'y avait rien à lire et la fonction read() renvoie 0.

Remarques :

- la fonction **read()** **ne place pas de caractère nul ('\0') après le dernier caractère lu;**
- S'il n'y a pas d'erreur de paramètre, **la fonction renvoie le nombre de caractères lus.** Dans le **cas d'un fichier associé à un terminal ou à un port série** :
  - ➢ Quand la voie de communication est configurée en mode canonique ou en mode non canonique avec MIN > 0, l'**appel de la fonction read() est bloquant** et la valeur qu'elle renvoie (le nombre de caractères lus) est toujours supérieur ou égal à 1.
  - ➢ Quand la voie de communication est configurée en mode non canonique avec MIN = 0, l'**appel de la fonction read()** est en quelque sorte **"bloquant pendant TIME*0.1 secondes"** et la valeur qu'elle renvoie (le nombre de caractères lus) peut être égal à 0.

Exemple :

```
char mesrecu [30];  int nblus;
nblus = read(fd, mesrecu, 29);          /* Lecture de 29 caractères au maximum */
if (nblus == -1)
{
        printf("Erreur lors de la lecture!\n");
}
else
{
        mesrecu[nblus] = '\0';          /* Ajout d'un '\0' après le dernier caractère lu */
        printf("Nombre de caractères lus : %d, message reçu : %s\n", res, mesrecu);
}
```

IV.5. Vidage des tampons
La fonction :
#include <termios.h>
int tcflush(int descripteur, int indicateur);

permet de vider un ou les 2 tampons de la voie communication entre le système et le terminal (ou le port série) associé au ***descripteur***, en fonction de la valeur de l'***indicateur*** :

Indicateur	Tampon(s) concerné(s)
TCIFLUSH	Tampon en entrée
TCOFLUSH	Tampon en sortie
TCIOFLUSH	Tampons en entrée et en sortie

IV.6. Fermeture du fichier
Un appel à la fonction :
#include <unistd.h>
int close (int descripteur);

ferme le fichier associé au ***descripteur*** et libère les ressources allouées lors de l'ouverture du fichier.

V. Echange de messages entre un poste émetteur et un poste récepteur (version 1)

On souhaite ici échanger des caractères entre un PC émetteur et un PC récepteur via une liaison RS232, ce qui implique que vous devez travailler en binôme.

Configuration matérielle de la liaison RS232 :
- vitesse de transmission = 19200 baud, avec une structure de trame 8E1;

Autres paramètres des voies de communication entre chaque système et son port série virtuel :
- **modes locaux :** mode non canonique sans echo, modes ISIG et ECHOE désactivés;
- **modes de contrôle :** modes HUPCL et CLOCAL actifs; mode CREAD actif coté récepteur;
- **modes d'entrée** : mode IGNBRK actif; modes IGNPAR, INPCK et PARMRK actifs afin de rejeter les caractères contenant une erreur de parité;
- **modes de sortie** : tous inactifs;
- MIN = 1 et TIME = 0.

Coté émetteur, le programme doit :
- ouvrir le fichier associé au port série virtuel;
- configurer la voie de communication entre le système et le port série conformément aux informations indiquées ci-dessus;
- demander à l'utilisateur de saisir les messages à envoyer, et les transmettre jusqu'à ce que l'utilisateur saisisse un mot ou un caractère (à choisir) qui indique qu'il souhaite quitter le programme;
- fermer le fichier associé au port série virtuel.

Coté récepteur, le programme doit :
- ouvrir le fichier associé au port série virtuel;
- configurer la voie de communication entre le système et le port série conformément aux informations indiquées ci-dessus;
- afficher convenablement les messages reçus;
- se terminer quand il reçoit à le mot ou le caractère (choisi à l'avance) indiquant la fin du programme émetteur, après avoir fermer le fichier associé au port série virtuel.

Remarque : vous devez **tester toutes les valeurs renvoyées par les fonctions** présentées dans la partie IV afin de détecter un éventuel problème.

VI. Echange de messages entre un poste émetteur et un poste récepteur (version 2)

Même problème que dans la partie V.
Mais cette fois, pour le résoudre, vous devez **déclarer, définir et utiliser** une **classe de communication** que l'on peut nommer **CComRS232**.

Cette classe doit permettre à ses utilisateurs, **de façon sécurisée et transparente*,** de :

- choisir le port série à utiliser;
- configurer la liaison série suivant les choix de l'utilisateur. Ce qu'il doit pouvoir choisir :
  - ✓ la vitesse de transmission;
  - ✓ le format de la trame;
  - ✓ les valeurs des paramètres MIN et TIME (le mode canonique sera toujours désactivé);
- utiliser la liaison série pour émettre et/ou recevoir des données.

* : les appels système réalisés dans les définitions des méthodes de la classe seront transparents pour l'utilisateur.

Pour cela, cette classe devra être muni :

- d'un constructeur et d'un destructeur;
- d'une méthode permettant d'émettre un message;
- d'une méthode permettant de recevoir un message;
- d'une méthode permettant de vider un ou les 2 tampons de la voie communication.

Remarques :

- les données doivent être correctement encapsulées;
- cette classe de communication doit être **sans saisie et sans affichage** (i.e. : les méthodes de la classe CComRS232 doivent être sans saisie et sans affichage).
- vous devez **tester toutes les valeurs renvoyées par les fonctions** présentées dans la partie IV afin de détecter un éventuel problème.
- Dans le fichier d'entête de la classe, vous devez **justifier soigneusement** :
  - le **type et le rôle de chaque attribut**;
  - le **prototype de chaque méthode** :
    - type et valeurs possibles de la valeur de retour;
    - type et rôle de chaque paramètre;

VII. Plan de tests de la classe CComRS232

Réaliser un plan de tests de la classe de communication.

VIII. Saisie d'un mot de passe

Par défaut, l'entrée standard, la sortie standard et la sortie d'erreur standard d'un processus sont **connectées à son terminal de contrôle**. De plus, un processus peut désigner son terminal de contrôle en utilisant la référence /dev/tty. En ouvrant ce fichier avec la fonction open(), un processus peut récupérer un descripteur associé à son terminal de contrôle.

Réaliser un programme qui demande à un utilisateur de saisir un mot de passe (le mot de passe ne devra pas s'afficher lors de la saisie), puis affiche ce mot de passe (pour prouver que le programme à réussi à l'acquérir !).

IX. Discussion entre 2 postes à la fois émetteur et récepteur

En utilisant votre classe de communication, réaliser un (ou deux) programme(s) permettant à 2 utilisateurs d'échanger des messages entre 2 PCs via une liaison RS232.