La gestion des ports séries sous Linux

Article pour l'Écho de Linux (***Juillet 1996***)
Eric Sellin (esellin@pratique.fr)

Sommaire

Introduction

L'ambition de cet article est bien modeste : il veut vous expliquer comment on accède aux ports séries sous Linux, c'est-à-dire comment y lire des caractères et en envoyer, comment modifier les paramètres de la liaison, etc...

Nous entrons tout de suite dans le vif du sujet avec quelques rappels élémentaires.

Bref rappel sur les liaisons séries

Sur une liaison série, les bits qui composent un caractère sont envoyés les uns aprés les autres, une tension étant associée aux 0 et aux 1. Les principaux paramètres d'une telle liaison sont les suivants :

- Son débit, exprimé en bits par seconde (bps)
- La parité (paire, impaire ou aucune)
- Le nombre de bits par caractère (5, 6, 7 ou 8)
- Le nombre de bits de stop (1, 1,5 ou 2)

Le débit indique combien de bits circulent chaque seconde sur la liaison. Cela peut aller de 50 à 460800 (!) bits/s.

La parité, lorsqu'elle est utilisée, est un bit supplémentaire qui sert de contrôle de transmission, contrôle qui est loin d'être parfait puisqu'il n'est efficace que si le nombre de bits érronés dans un caractère est impair.

Chaque caractère peut être composé de 5 à 8 bits. Dans la grande majorité des cas, on utilise 8 bits.

Le ou les bits de stop servent à marquer la fin de chaque caractère dans la transmission. En général, on n'utilise qu'un seul bit de stop.

Ouverture de la liaison

Sous Linux, chaque port série de votre machine est représenté par un *fichier de périphérique* situé dans le répertoire /dev à côté de dizaines d'autres.

- /dev/ttyS0 correspond à COM1
- /dev/ttyS1 correspond à COM2
- /dev/ttyS2 correspond à COM3
- /dev/ttyS3 correspond à COM4

Au niveau d'un programme en C, on ouvre ces fichiers exactement comme on ouvrirait n'importe quel autre fichier grâce à l'appel système *open()* :

```
int fd;
if ( (fd=open("/dev/ttyS1",O_RDWR)) == -1 ) {
        perror("open");
        exit(-1);
}
```

Une fois que le port est ouvert, on peut y lire et y écrire des caractères au moyen des primitives *read()* et *write()*.

Mais ne nous emballons pas, il faut d'abord paramétrer notre liaison.

Paramétrage de la liaison

Dans la norme POSIX, tous les paramètres d'une liaison sont regroupés dans une structure appelée *termios* et définie dans le fichier <termios.h> qu'il nous faut donc inclure.

Cette structure comporte les champs suivants

```
struct termios {
        tcflag_t c_iflag;               /* input mode flags */
        tcflag_t c_oflag;               /* output mode flags */
        tcflag_t c_cflag;               /* control mode flags */
        tcflag_t c_lflag;               /* local mode flags */
        cc_t c_line;                    /* line discipline */
```

```
        cc_t c_cc[NCCS];                /* control characters */
};
```

La fonction *tcgetattr* permet d'obtenir les paramètres actuels d'une liaison. En voici le prototype, extrait du *man* :

```
int tcgetattr ( int fd, struct termios *termios_p );
```

et un exemple typique d'utilisation :

```
struct termios termios_p;
tcgetattr(fd,&termios_p);
```

Notre structure *termios_p* étant renseignée, on peut en modifier les champs, dont une description sera donnée au paragraphe suivant.

Une fois que ces champs ont été modifiés, il faut enregistrer ces modifications au moyen de la fonction *tcsetattr* :

```
tcsetattr(fd,TCSANOW,&termios_p);
```

Les champs de la structure termios

Nous n'allons pas détailler ici l'ensemble des champs de cette structure car ils sont trop nombreux. Seuls les champs utiles seront abordés.

- c_iflag : les modes d'entrée

  Ils définissent un traitement à appliquer sur les caractères en provenance de la liaison série :
  - IGNBRK : les caractères BREAK sont ignorés.
  - IGNPAR : les caractères qui comportent une erreur de parité sont ignorés.
  - ISTRIP : dans le cas d'une transmission sur 8 bits, le huitième bit est systématiquement mis à zéro.

- c_oflag : les modes de sortie

  Ils définissent un traitement à appliquer sur les caractères envoyés sur la liaison série. On y trouve notamment OLCUC qui transforme les minuscules (non accentuées !) en majuscules.

- c_cflag : Les modes de contrôle

  Ce champ est important, car c'est ici que l'on définit le débit, la parité utilisée, les bits de donnée et de stop, ...
  - CLOCAL : si ce flag n'est **pas** utilisé, une déconnexion (ie. chute du signal DCD) entraîne la fermeture automatique du port série et les prochains appels à *read()* renverront la valeur -1.
  - CS5 : chaque caractère est composé de 5 bits.
  - CS6 : chaque caractère est composé de 6 bits.
  - CS7 : chaque caractère est composé de 7 bits.
  - CS8 : chaque caractère est composé de 8 bits (c'est le cas le plus fréquent).

- CSTOPB : on utilise 2 bits de stop au lieu d'un seul par défaut (sauf en 5 bits, où on utilise 1,5 bits).
- HUPCL (Hang UP on CLose) : provoque la déconnexion (ie. chute des signaux DTR et RTS) lors de la fermeture du port série par l'appel système *close()*.
- PARENB : activation du mécanisme de parité paire.
- PARODD : utilisation d'une parité impaire au lieu d'une parité paire par défaut.
- B50 : liaison à 50 bits/s.
- B75 : liaison à 75 bits/s.
- ...
- 230400 : liaison à 230400 bits/s.
- 460800 : liaison à 460800 bits/s.

- c_lflag : les modes locaux

  Ce champ est d'une importance capitale : il définit le mode (canonique ou non) et la gestion de l'écho.
  - ECHO : un écho des caractères reçus est effectué.
  - ICANON : passage en mode canonique, c'est-à-dire que les caractèes reçus sont stockés dans un tampon et qu'ils ne sont disponibles qu'à la réception d'un caractère *eol* (de code ASCII décimal 10). Les caractères *erase* et *kill* permettent respectivement d'effacer le dernier caractère et tous les caractères de la ligne courante. En mode non-canonique (le mode par défaut), tous les caractères sont immédiatement disponibles à la lecture.

- c_cc : les caractères de contrôle

  C'est un tableau de caractères. Des constantes sont définies pour accéder à quelques éléments particuliers :
  - VERASE : en mode canonique, c'est le caractère *erase*, qui provoque l'effacement du dernier caractère.
  - VKILL : en mode canonique, c'est le caractère *kill*, qui provoque l'effacement de toute la ligne.
  - VEOF : en mode canonique, c'est le caractère de fin de fichier lorsqu'il est utilisé seul sur une ligne. Le prochain appel à *read()* renverra une valeur nulle.
  - VEOL : en mode canonique, c'est le caractère de fin de ligne. C'est lui qui rend disponibles à la lecture tous les caractères précédents.
  - VMIN : en mode non-canonique, spécifie le nombre de caractéres que doit contenir le tampon pour être accessible à la lecture. En général, on fixe cette valeur à 1.
  - VTIME : en mode non-canonique, spécifie, en dixièmes de seconde, le temps au bout duquel un caractère devient accessible, même si le tampon ne contient pas *c_cc[VMIN]* caractères. Une valeur de 0 représente un temps infini.

Mode bloquant et mode non-bloquant

Que se passe-t-il lorsqu'on appelle *read()*, et qu'il n'y a malheureusement rien à lire ?

Deux possibilités s'offrent à nous :

- l’appel *read()* reste en attente d’un caractère. C’est le fonctionnement par défaut.
- l’appel *read()* se termine immédiatement et retourne la valeur -1.

C’est à vous de choisir le mode de fonctionnement. Vous pouvez faire ce choix à l’ouverture du port série, mais également à n’importe quel autre moment.

À l’ouverture par *open()*, utilisez l’indicateur O_NONBLOCK :

```
int fd;
if ( (fd=open("/dev/ttyS1",O_RDWR|O_NONBLOCK)) == -1 ) {
        perror("open");
        exit(-1);
}
```

Pour modifier le fonctionnement alors que le port série est déjà ouvert, il faut utiliser l’appel système *fcntl()* de la façon suivante :

```
/* Passe en mode bloquant */
fcntl(fd,F_SETFL,fcntl(fd,F_GETFL)&~O_NONBLOCK);

/* Repasse en mode non-bloquant */
fcntl(fd,F_SETFL,fcntl(fd,F_GETFL)|O_NONBLOCK);
```

Exploitation

Je vous rappelle qu’on peut lire et écrire avec les appels systèmes *read()* et *write()*.

Voici quelques informations pour mieux exploiter votre liaison :

- Lorsque vous effectuez un appel à *write()*, les caractères que vous voulez envoyer sont placés dans un tampon et *write()* rend la main immédiatement. Votre programme continue donc de s’éxécuter pendant que des caractères sont envoyés sur le port série par le noyau. Pour arrêter brutalement l’envoi, utilisez *tcflush()* de la manière suivante :

  ```
  tcflush(fd,TCOFLUSH);
  ```

- De même, pour éliminer tous les caractères reçus par le noyau mais non encore lus par *read()*, faites ceci :

  ```
  tcflush(fd,TCIFLUSH);
  ```

- Pour bloquer le processus jusqu’à ce que tous les caractères aient effectivement été envoyés, utilisez *tcdrain* comme suit :

  ```
  tcdrain(fd);
  ```

- Lorsque vous êtes en mode non-bloquant, pour attendre l’arrivée d’un caractère, n’appelez surtout pas *read()* jusqu’à ce qu’il vous renvoie une valeur différente de -1. Cela charge inutilement votre machine. Utilisez plutôt l’appel système *select()*.

- Dans le cas d’une utilisation avec un modem, pour faire baisser le signal DTR et ainsi provoquer la déconnexion, il suffit de passer à une vitesse nulle en utilisant la constante de vitesse B0.

Programme d'exemple

Voilà, on est loin d'avoir tout dit sur le sujet et vous pouvez bien sûr consulter *man termios* si vous voulez plus d'informations. Mais ça nous suffit largement pour aborder tout de suite un petit programme d'exemple :

```
#include <stdio.h>
#include <termios.h>
#include <sys/fcntl.h>

void main(void)
{
        int             fd;
        char            c;
        struct termios  termios_p;

        /* Ouverture de la liaison serie */
        if ( (fd=open("/dev/ttyS1",O_RDWR)) == -1 ) {
                perror("open");
                exit(-1);
        }

        /* Lecture des parametres courants */
        tcgetattr(fd,&termios_p);
        /* On ignore les BREAK et les caracteres avec erreurs de parite */
        termios_p.c_iflag = IGNBRK | IGNPAR;
        /* Pas de mode de sortie particulier */
        termios_p.c_oflag = 0;
        /* Liaison a 9600 bps avec 7 bits de donnees et une parite paire */
        termios_p.c_cflag = B9600 | CS7 | PARENB;
        /* Mode non-canonique avec echo */
        termios_p.c_lflag = ECHO;
        /* Caracteres immediatement disponibles */
        termios_p.c_cc[VMIN] = 1;
        termios_p.c_cc[VTIME] = 0;
        /* Sauvegarde des nouveaux parametres */
        tcsetattr(fd,TCSANOW,&termios_p);

        /* Affichage sur le terminal */
        write(fd,"Tapez Ctrl-C pour quitter\n",26);

        /* Boucle de lecture */
        while ( 1 ) {
                read(fd,&c,1);
                if ( c == 0x03 )                /* Ctrl-C */
                        break;
                printf("%03u %02x %c\n",c&0xff,c&0xff,c);
        }

        /* Fermeture */
        close(fd);

        /* Bye... */
        exit(0);
```

```
}
```

Bibliographie

La programmation sous UNIX de Jean-Marie RIFFLET aux éditions Ediscience, excellent ouvrage que je vous recommande chaudement ainsi d'ailleurs que **La communication sous UNIX** du même auteur.