Périphériques série asynchrones pour les roboticiens

Matthieu Herrb
CNRS-LAAS

Introduction

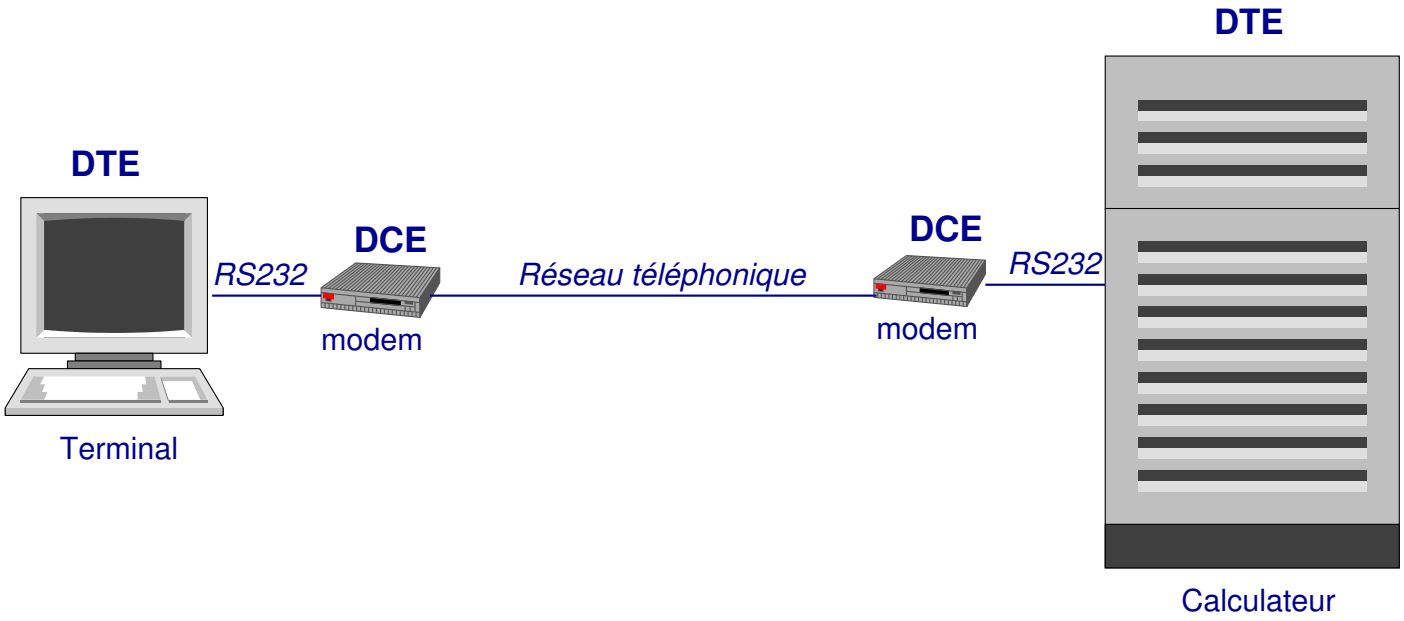
Lignes série asynchrones : RS232
- Standard « universel »
- Assez simple en apparence
- Nombreux défauts sources de difficultés sans fin

Mais plein d'autres supports font aussi des liaisons série asynchrones : USB, Ethernet, CAN, etc.

Plan :
- Rappels sur RS232
- Gestion des terminaux et des lignes séries sous Unix
- Communication avec des capteurs et des actionneurs
- Problèmes et solutions
- Conclusion

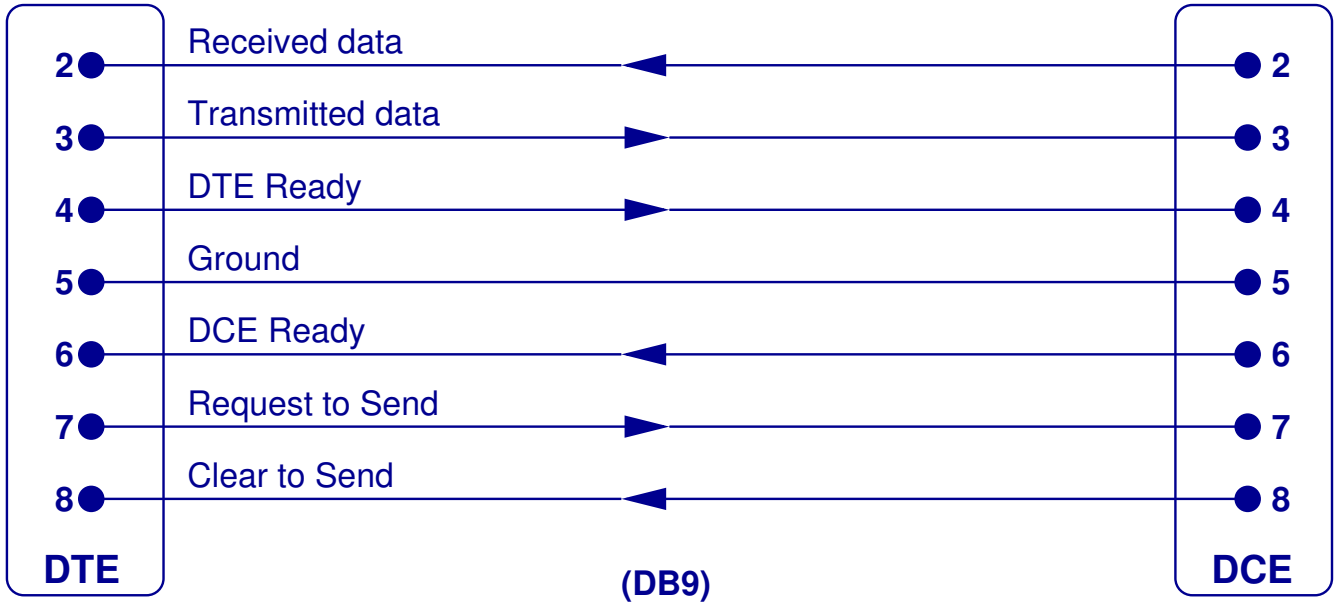
Rappels sur RS/232

- Introduit en 1962
- Communication full-duplex sur deux fils plus une masse commune.
- Lignes de «handshake» en plus
- Signaux indiqués par une tension : **ON** : $+3 \cdots +12$V **OFF** $-3 \cdots -12$V
- **DTE** (Data Terminal Equipement) : calculateur ou terminal
- **DCE** (Data Circuit-terminating Equipement) : Modem
- connecteurs **DB25**, **DB9** ou **RJ45**.



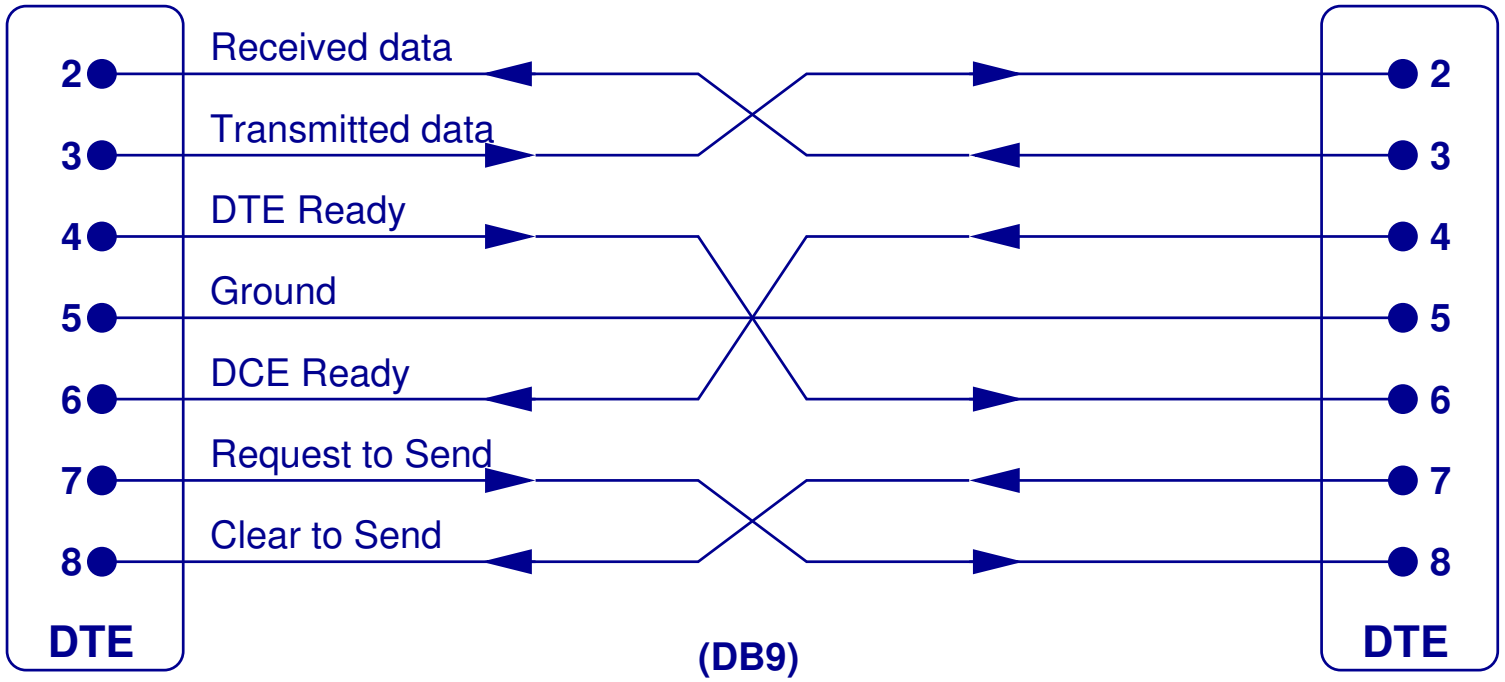
Signaux RS/232

TxD	Transmitted Data	
RxD	Received Data	
CTS	Clear To Send	DCE→DTE
DSR	Data Set Ready	DCE→DTE
DTR	Data Terminal Ready	DTE→DCE
RTS	Request to Send	DTE→DCE



Câbles Null-Modem

Connexion directe DTE/DTE :



Attention : il existe aussi des câbles null-modem qui rebouclent localement les fils de contrôle.

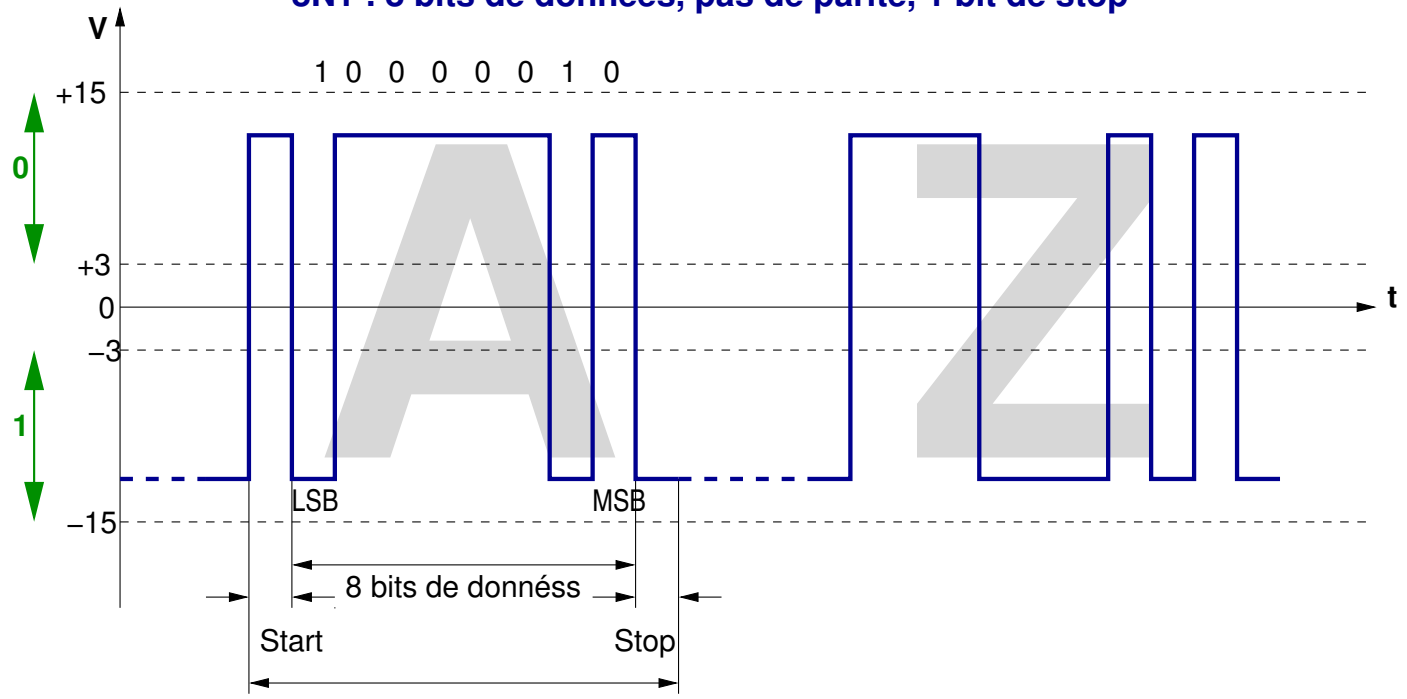
Certains équipements DTE incluent une conversion null-modem.

Structure d'un caractère

- un bit **start** (0)
- 7 ou 8 bits de données, faible poids en premier
- un ou deux bits **stop** (1)

Horloge donnée par la vitesse de transmission : 300, 1200, 4800, 9600, 19200, 38400, 57600, 115200 bauds.

8N1 : 8 bits de données, pas de parité, 1 bit de stop



Conventions

(Pas toujours respectées → source de problèmes sans fin)

- **DTE** : connecteur DB9 mâle ou DB25 femelle
- **DCE** : connecteur DB9 femelle ou DB25 mâle

Convention LAAS (P. Ribes) :
- Un câble long est toujours mâle-femelle et direct
- Un câble court mâle-femelle est toujours un câble null-modem
- Un câble court avec 2 prises du même sexe est toujours direct

→ Combiner plusieurs câbles pour construire l'adaptateur dont on a besoin.

→ Toujours étiqueter clairement les câbles qui n'obéissent pas à cette convention !

Gestion des ports série dans Unix

Utilisation principale : connexion des terminaux (télétype – TTY)

`/dev/tty*`

Mode de fonctionnement par défaut dit «*cooked*» :
- orienté ligne : transmet les données ligne par ligne (sur le caractère «CR»).
- possibilité d'édition locale : erase char(`^H`), erase word(`^W`), erase line(`^U`)
- `^C` envoie le signal SIGINT aux processus attachés à ce terminal
- `^\` envoie le signal SIGQUIT aux processus attachés à ce terminal
- écho des caractères transmis par le terminal
- remplacement du caractère CR dans l'entrée par LF
- remplacement du caractère LF en sortie par la paire CR/LF

On retrouve ce mode émulé dans les terminaux virtuels utilisés par les systèmes multi-fenêtres (X window : `xterm`).

Pas adapté à la connexion d'autre chose qu'un terminal.

Ports série sur le matériel

PC Linux

COM1 `/dev/ttyS0`
COM2 `/dev/ttyS1`

Autres ports série, nom du périphérique dépend du driver.
Exemple : carte Rocketport (robots iRobot) : `/dev/ttyR0`...`/dev/ttyR7`.

Application pour émuler un terminal : **minicom**.

Sun Solaris

`/dev/term/a`
`/dev/term/b`

Application pour émuler un terminal : **tip**.

Programmation des modes d'un terminal : TERMIOS

Interface définie par la norme POSIX.

```
#include <termios.h>
#include <unistd.h>

int tcgetattr(int fd, struct termios *termios_p);
int tcsetattr(int fd, int optional_actions, struct termios *termios_p);

int tcsendbreak(int fd, int duration);

int tcdrain(int fd);
int tcflush(int fd, int queue_selector);

int tcflow(int fd, int action);

int cfmakeraw(struct termios *termios_p);
speed_t cfgetispeed(struct termios *termios_p);
speed_t cfgetospeed(struct termios *termios_p);
int cfsetspeed(struct termios *termios_p, speed_t speed);
```

La structure termios

Contient au moins :

```
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 */
```

`cfmakeraw()` positionne tous les champs pour passer en mode «raw» :

- communication caractère par caractère (Voir plus loin)
- aucune interprétation de caractères spéciaux
- aucun écho

Voir la page de manuel `termios(7)` pour plus de détails.

Un exemple simpliste

```c
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <termios.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int fd, n;
    struct termios tio;
    char c;

    fd = open("/dev/ttyS0", O_RDWR);
    tcgetattr(fd, &tio);
    cfmakeraw(&tio);
    cfsetspeed(&tio, B9600);
    tcsetattr(fd, TCSAFLUSH, &tio);

    for (;;) {
        do {
            n = read(fd, &c, 1);
            if (n < 0 && errno == EINTR)
                continue;
        } while (n <= 0);
        printf("recu : %c\n", c);
    }
}
```

Lecture bloquante ou non ?

Termios permet de paramétrer le comportement de l'appel système **read()** lorsqu'il n'y a rien à lire.

Deux paramètres de termios contrôlent le comportement de `read()` :

VMIN $>$ 0, VTIME $>$ 0 : attend au moins un caractère, puis au plus VTIME/10 secondes pour recevoir au moins VMIN caractères.

VMIN $>$ 0, VTIME $=$ 0 : attend au moins VMIN caractères.

VMIN $=$ 0, VTIME $>$ 0 : attend au plus VTIME/10 secondes pour un caractère.

VMIN $=$ 0, VTIME $=$ 0 : retourne les caractères disponibles, sans bloquer.

Par défaut `cfmakeraw()` définit VMIN=1 VTIME=0.

La réception d'un signal interrompt `read()` qui retourne alors -1 et positionne `errno` à **EINTR**

Voir aussi `fcntl(O_NONBLOCK)` et **SIGIO**...

Les périphériques utilisés en robotique

Nombreux capteurs ou actionneurs utilisent une interface série :
- RS/232
- USB (à travers un convertisseur USB/Série ou pas)
- ...

Divers protocoles d'échange, ASCII ou binaire, mais des points communs :
- Le périphérique envoie des données, et reçoit des commandes,
- La communication est asynchrone,

Inversion des rôles :
- Le périphérique se comporte comme le calculateur.
- L'interface de la couche fonctionnelle (module GenoM) émule l'utilisateur d'un terminal.

Catalogue

Quelques appareils avec liaisons asynchrones utilisés sur les robots du LAAS :

compas magnétiques : lama, diligent,

gyroscopes : hilare 2, lama, rackham

centrales inertielles : adam, lama, kharma

platines pan/tilt : rackham, lama, dala, jido (*platine*), cameras Sony pan/tilt : diligent, rackham

moteurs CC : lapa, jido

micro-controleurs d'un robot complet : diligent (*xr4000*), dala, rackham (*rflex*), karma, Lhassa

GPS : karma (*gpssmu*), lhassa, dala

Télémètres laser SICK : hilare2, diligent, dala, rackham, jido,... (*sick*)

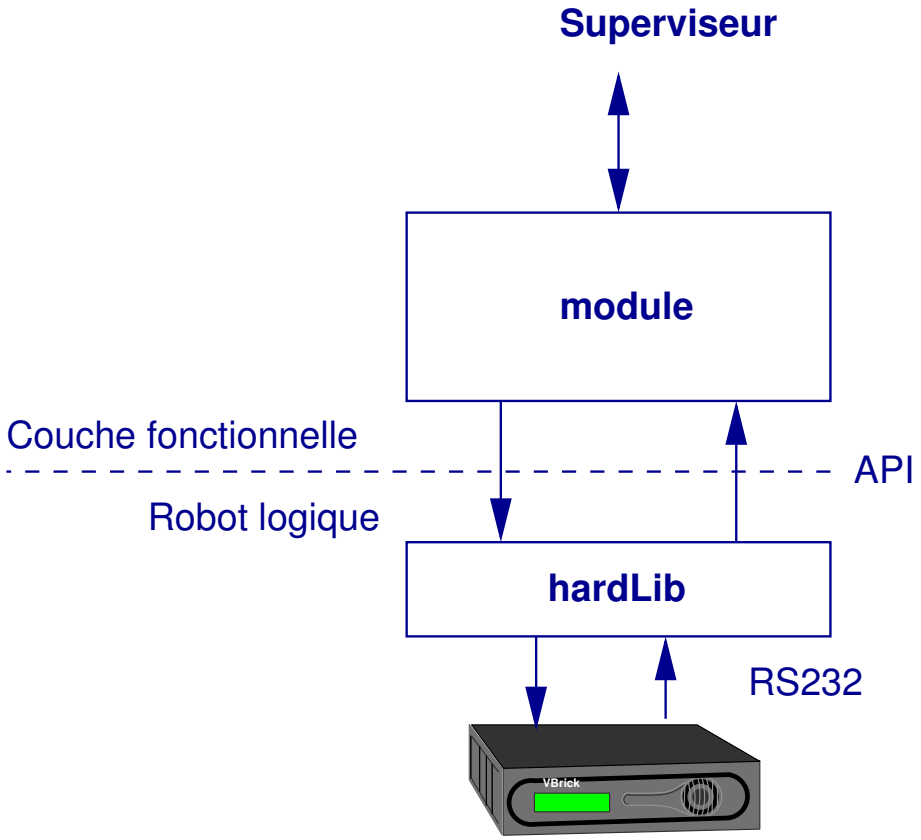
Problèmes potentiels

- périphérique = automate à états : connaître son état et rester synchronisés,
- perte possible de caractères : désynchronisation de l'état,
- identifier de manière robuste et non ambiguë le début et la fin des messages,
- taille du plus grand message à lire pour se resynchroniser,
- effets des buffers de réception et d'émission,
- 2 automates asynchrones inter-connectés : risque d'inter-blocage,
- datation des données reçues.

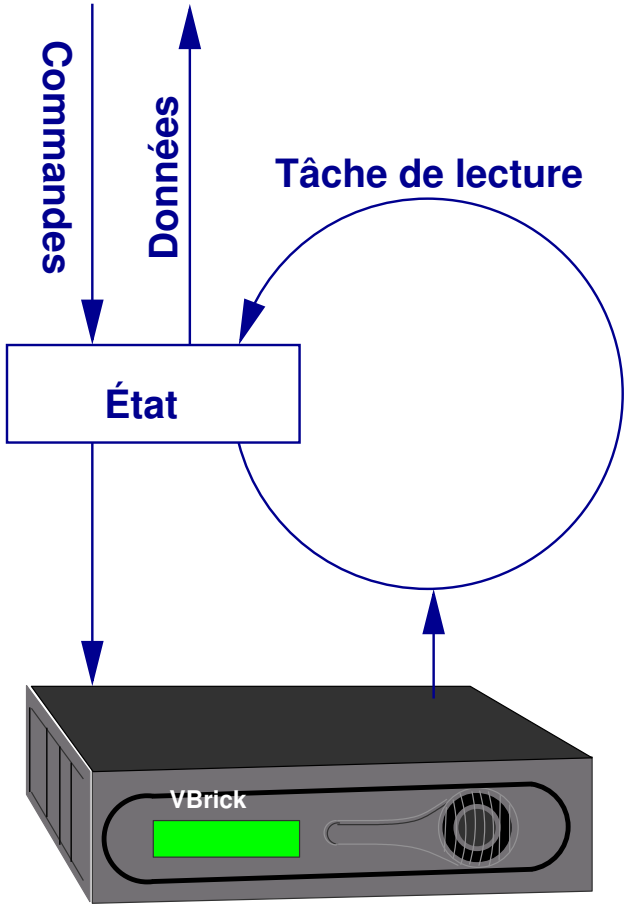
Méthodologie

Créer une **interface de programmation** qui masque ces problèmes aux utilisateurs :

- toutes les fonctions retournent rapidement
- quelques fonctions bloquantes pour attendre un évènement pour tests, (mais pas utilisées par GenoM)
- fonctions ré-entrantes pour pouvoir gérer plusieurs périphériques similaires dans le même module
- gestion de l'asynchronisme et de l'état en interne (souvent avec une tâche séparée),
- exporter les données utiles vers la couche fonctionnelle, convertir les unités.



Structure de la bibliothèque



Outils

- **fonctions termios de base** dans de nombreuses bibliothèques, plus ou moins ad-hoc, à adapter...
- **buffers circulaires** `hardLib/serial.c` (trop compliqué)
- **tâche de lecture/décodage** ah-hoc à chaque fois.
  Exemple pas trop compliqué dans `fh2805`.
  Exemple plus complet dans `platine`.

Conclusion

Communications série asynchrones : très répandues.

Résoudre les problèmes de câblage.

Programmation des modes sous Unix via **termios**

Créer une **tâche de lecture** séparée pour maintenir l'état du périphérique et éviter les inter-blocages.