CUCCARONI Claire
3A – SERTR

Projet de fin d'étude

Autour des systèmes embarqués :

Module Fox Box d'Acmesystems

Tuteurs : Pierre TRICOT
Michel DUFNER

Présentation du sujet

A partir d'une plate-forme de type PC-embarqué, comme l'association du module FOX Board et d'un PC, on souhaite développer un ensemble de stratégies d'utilisations ainsi qu'une bibliothèque de fonctions utilisables de ce cadre. A terme, on souhaite construire un système complet permettant la communication entre le module embarqué et un terminal distant par Ethernet, ce système pouvant être exporté sur d'autre cibles que la carte FOX comme la plate-forme SNAP (IMSYS).

Abstract

The purpose of the present study is to develop a ensemble of strategy of uses of a PC-embedded platform, like Fox Board, and a library of useful functions. Finally, we want to build a whole system, which enables the communication between the embedded platform and a distant terminal via an Ethernet connection. Moreover, this system would be export on other targets than FOX Board, like SNAP platform.

Sommaire

INTRODUCTION

Le monde des cartes embarquées est actuellement en plein essor. En effet, ces modules gèrent un grand nombre de systèmes du plus simple au plus compliqué, de la machine à café à la gestion du freinage d'une voiture. Toutefois, un grand nombre de plates-formes embarquées sont conçues pour un unique système, répondant à des besoins et donc à des cahiers des charges précis. Une des exceptions à cette règle : la Fox Board d'Acme Systems.

Le but de cette étude est de réaliser une application gérant les entrées/sorties de la Fox Board, exportable sur une plate-forme du même type, la SNAP d'IMSYS.
Pour cela, j'ai tout d'abord travaillé sur l'installation et le développement des cartes Fox. Puis, une fois le module SNAP disponible, je me suis concentrée sur son étude et sur son système de développement.

Dans ce rapport, je présente en premier lieu les deux types de cartes embarquées Fox et SNAP. Puis, j'explique le travail réalisé ainsi que le cheminement que j'ai suivi pour se faire.

I. Présentation des modules

A) Module Fox Board

1. Carte Fox

Le module FOX Board supporte, malgré sa petite taille (66 mm*72 mm), un véritable système Linux, ce qui lui permet d'être utilisée pour des tâches variées comme un grand nombre de dispositifs réseaux allant du routeur au serveur Web ou encore en amélioration du simple micro-processeur d'une application embarquée. Elle dispose de plus d'une grand nombre de moyens de connexion : un port Ethernet 10/100, 2 ports USB 1.1, ainsi que 2 groupes de 40 bornes permettant l'ajout de d'interfaces ou la connexion avec d'autres modules ou applications. On peut ainsi connecter le module Fox à une platine d'interface comme la SX18 de Lextronics.

Illustration 1: Module Fox Board

2. Platine d'interface SX18

La plate-forme SX18 et la carte Fox se connectent grâce aux 40 bornes libres du module, qui sont justement réservées à ce genre d'extension. La plate-forme SX18 dispose entre autres d'un port série RS232, de boutons poussoirs et de LEDs. Elle est équipée d'un étage de régulation qui permet d'alimenter l'ensemble sous une tension pouvant être comprise entre 7 et 15 Vcc.

Illustration 2: Platine d'interface SX18 (avec module Fox)

B) Module SNAP Simple Application Network Platform

1. Carte SNAP

Le module SNAP (Simple Application Network Platform) est un module équipé d'un microcontrôleur développé par Imsys, utilisé essentiellement pour le développement de systèmes embarqués équipés de connexion LAN ou Internet.

Le microcontrôleur abrite l'environnement d'exécution Java 2 Micro Edition-CLDC qui permet l'exécution de Java code sans machine virtuelle, ce qui lui permet d'être plus rapide que les produits similaires.

Illustration 3: Module SNAP

Le module SNAP dispose d'un certain nombre d'interfaces permettant la gestion de périphériques standards et d'une possibilité de connexion Ethernet avec un serveur TCP/IP entièrement implémenté.

Pour pouvoir l'utiliser en développement, le module SNAP fonctionne en combinaison avec une carte-mère et un PC. Une carte mère (motherboard) est une plate-forme munie d'entrées-sorties dans laquelle il est possible de monter le module SNAP. Les plus simples offrent les ports courants (Ethernet, RS-232...), les plus sophistiquées peuvent également présenté écrans ou claviers pour faciliter l'utilisation du module.

2. Carte-mère TutorIO

Un exemple de carte-mère est la TutorIO de Taylec. Elle dispose d'un port Ethernet et de 2 ports série standard. Elle comprend également 8 outputs et 8 Inputs digitaux, qui peuvent être utilisés par à l'aide d'une nappe par un appareil externe afin de commander d'autres appareils. Un convertisseur analogique-numérique (8-bit DAC et 8-bit ADC) se trouve sur la carte mère afin de permettre la communication avec la monde analogique. Les interfaces 1-Wire, CAN bus, ADC input, DAC output et I2C peuvent être utilisées par des borniers à vis.

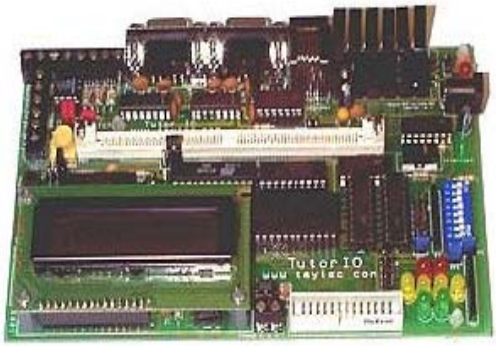

Illustration 4: Carte-mère TutorIO

II. Étude réalisée

A) étude carte Fox

1. Prise en main

- Installation

On a dans un premier temps cherché à prendre en main la carte Fox.
Pour cela, il a fallu mettre en place le système de développement (SDK). Le SDK fourni par Acmesystems n'étant pas compatible Windows, deux solutions étaient alors possibles : soit travailler sous Windows en utilisant un émulateur Unix comme VMware, soit placer la station de développement directement sous Linux. C'est cette dernière solution qui a été privilégiée. Elle permettait en effet d'éviter les problèmes de compatibilité et de droits lors des compilations, ainsi qu'un éventuel ralentissement du SDK dû à l'émulateur.

Le développement requérant des droits d'administrateurs, on a décidé de réserver des machines aux projets devant utiliser les modules Fox Board. Trois projets ayant originellement été prévus, trois machines ont été installées avec un environnement Linux Debian. La distribution Debian a été choisie parmi les distributions possibles car lors du début du projet le SDK n'était compatible directement qu'avec des distributions Debian ou RedHat. Depuis, Acmesystems a élargi son champ d'actions et le SDK est maintenant compatible avec d'autres distributions comme Ubuntu, qui peuvent être plus faciles à utiliser lors d'une première approche avec Linux.

Les machines ont été placées en réseau au moyen d'un hub. Les cartes Fox ont également été configurées pour pouvoir être connectées au même réseau. En effet, à la livraison le module Fox a une adresse IP par défaut, qu'il faut donc modifier pour éviter les incohérences si 2 cartes sont connectées en même temps.

Pour permettre aux machines de communiquer à la fois avec le réseau local, i.e. avec les cartes Fox, et avec l'extérieur par Internet, on avait 2 solutions possibles : soit ajouter des cartes réseaux aux machines, soit implanter des interfaces virtuelles. C'est cette seconde solution qui a été retenue pour des raisons évidentes de coût et de délai. La création d'une seconde interface sur la carte réseau d'un ordinateur lui permet de se comporter comme s'il disposait d'une seconde carte réseau. L'interface par défaut a ainsi été configurée en dhcp pour permettre à l'ordinateur de

communiquer avec l'extérieur via le réseau et le proxy de l'école. La seconde interface a été configurée avec une adresse statique permettant la connexion au réseau local 192.168.0.0/24.

L'installation des environnements Debian et du SDK sur les trois machines a demandé un certain temps. De même la simple prise en main de la distribution Debian, dont le fonctionnement en administrateur m'était quasiment inconnu, a nécessité un temps d'adaptation.

➢ Fonctionnement du SDK

L'utilisation du SDK est assez simple à condition de savoir exactement ce que l'on veut faire. Le SDK est essentiellement constitué du cross-compiler (compilateur croisé) permettant la compilation de programmes (écrits en C) sur la machine de développement. Normalement, la compilation simple d'un programme construit un exécutable utilisable uniquement sur la machine sur laquelle on compile. Dans le cas d'un module externe comme la Fox Board, la compilation doit être réalisée sur la machine de développement (où se trouve le programme et le compilateur) de façon à pouvoir fonctionner sur le module où l'exécutable est ensuite transféré.

La compilation est lancé dans une invite de commande ou terminal. (commandes en annexe 1.1)
D'abord il faut lancer l'initialisation des variables d'environnement de façon à ce que lors de la compilation, le compilateur trouver toutes les librairies nécessaires.
Ensuite on réalise la compilation proprement dite. La première étape dépend du type de librairie utilisée pour la compilation (glibc ou uCLibc) et crée dans le répertoire un fichier caché qui contient les instructions pour les librairies et l'environnement à utiliser dans la compilation du programme. Puis on lance la compilation.

L'exécutable ainsi obtenu doit ensuite être transféré dans le module Fox Board au moyen par exemple d'un protocole scp.

Pour bien comprendre le fonctionnement du SDK, j'ai réalisé l'application proposée en exemple : l'application "hello world". Cette application est l'application classique pour les débutants dans un grand nombre de systèmes. Elle consiste à faire dire "hello" au programme. Le programme, très simple, est fourni par Acmesystems ainsi que son makefile. (cf annexe 1.2) J'ai donc compilé

le programme et transféré l'exécutable sur un module Fox via un protocole scp, puis testé le bon fonctionnement de ce programme en me connectant au module par Telnet.

J'ai ensuite réalisé quelques programmes simples comme un programme "perroquet" qui répète les caractères tapés au clavier.

2. Connexion d'un périphérique : carte son USB

Pour les besoins d'un autre projet, nous avons tenté de connecter sur la carte fox une carte son USB. Le but de cette manipulation était de réaliser l'acquisition et la restitution de son pour la réalisation d'une application de Voix sur IP.

Tout d'abord, comme l'environnement embarqué dans la Fox Board est un environnement Linux, nous avons vérifié que ce périphérique était bien reconnu sous Linux. En connectant cette carte son USB à un PC sous Linux, on a constaté que c'était bien le cas.
Comme en la connectant directement sur un port USB du module Fox, celui-ci ne reconnaissait pas la carte son USB, nous avons recherché un driver qui comblerait cette lacune. Nous avons ainsi trouvé un driver alsa correspondant aux références obtenues lors de la connexion de la carte son avec un PC.

L'étape suivante a consisté à installer le driver sur le module Fox. En étudiant la documentation d'Acmesystems, il est apparu qu'il était possible d'installer différents périphériques comme par exemple une Webcam. Pour cela, il fallait réaliser la compilation du driver de façon à le rendre exécutable par le module. En récupérant les makefiles proposés pour les exemples comme la Webcam par le site d'Acmesystems, on a tenté de ré-écrire un makefile compatible avec le driver de la carte son USB. Toutefois, il semble que le langage utilisé par le Linux du module Fox diffère de celui d'un noyau classique, du fait sans doute qu'il est installé dans un plus petit espace et qu'il a été adapté aux besoins du module. De plus, la construction d'un makefile pour un driver est d'un niveau autrement plus compliqué que celui requis pour un simple programme. En conséquence de quoi et malgré le temps passé sur le sujet, la connexion de la carte son USB n'a pu être menée à bien. Le projet parallèle de VoIP qui avait initié cette recherche a utilisé un autre moyen pour réaliser les acquisition et restitution de son.

3. Entrées/sorties

Communication par Ethernet

La communication de base entre le module Fox et le PC de développement est réalisé via le port Ethernet par des protocoles comme scp (pour le transfert de programmes) ou Telnet (pour travailler directement en tant qu'administrateur du Linux embarqué) Les directives à suivre sont alors les mêmes que pour réaliser ce type de connexion avec un autre PC.

Lignes I/O

Le module Fox dispose de 3 bus de communication : 2 bus de 8 bits (A et B) et un bus de 32 bits (G). A chaque bit de ces bus correspond une ou plusieurs connexions physiques parmi les 40 bornes des 2 borniers d'extension. Un certain nombre de ces lignes sont réservées à un usage précis comme par exemple la communication avec un port série. Les autres sont d'un usage général et peuvent être selon les lignes soit uniquement en entrée, soit uniquement en sortie, soit en entrée/sortie avec un sens par défaut. La direction des ces dernières lignes peut être affectée bit par bit pour certaines lignes et doit être affectée par octet (8 lignes en même temps) pour d'autres. Certaines lignes en entrée/sortie peuvent être placées sur 2 liaisons physiques, en entrée sur l'une et en sortie sur l'autre. Ces lignes d'entrées/sorties sont connectées lors de la mise en place de la carte Fox dans la platine d'interface SX18.

La documentation concernant la carte-mère SX18 étant très restreinte, le principal problème lors de son installation a été de trouver quelle ligne I/O du module Fox correspond à quel composant de la carte-mère.
Par exemple, sur les 3 LEDs situées sur la carte-mère et censément en communication avec le module, une seule a été formellement identifiée comme reliée à la ligne de sortie OG25 (bit n°25 du port G). De même, parmi les 40 bornes servant à la communication entre la carte-mère et la carte Fox, la documentation d'Acmesystems recense 2 groupes d'I/O ttyS2 et ttyS3, qui pourraient tous deux être utilisés pour la communication par un port série. Toutefois, un seul port série RS232 n'est présent sur la SX18. Il a donc fallu faire un certain nombre d'essais pour déterminer que le port RS232 de la SX18 correspondait au périphérique ttyS3 de la carte Fox.
A l'heure actuelle, les demandes de documentation effectuées par mail auprès de Lextronics n'ont eu aucune réponse.

4. Application d'entrées/sorties

➢ Communications tout ou rien

On souhaite réaliser différents types d'entrée/sortie avec le module Fox. On s'est d'abord attaché aux communications tout ou rien (sur 1 bit).
Ce type d'entrée/sortie correspond à faire passer à 1 ou à 0 un seul bit. Le module Fox offre pour cette manoeuvre deux fonctions.
La première méthode consiste à ouvrir le port en début de manipulation et à le refermer à la fin. Il faut ensuite définir un masque, en plaçant à 1 le bit correspondant au bit qu'on souhaite utiliser et à 0 les autres. On manipule ensuite le bit par la fonction ioctl() qui permet selon la syntaxe de placer à 0 ou 1 le bit ou contraire de récupérer la valeur de ce bit.
La seconde méthode utilise un ensemble de fonctions stockées dans la librairie gpio_syscalls, fournie uniquement avec la dernière version du système de développement. Il n'est pas nécessaire d'ouvrir ou de fermer les ports, ce qui accélère grandement le traitement des instructions.

Pour tester ces fonctions, on a réalisé deux programmes faisant clignoter une LED selon les 2 méthodes. (cf annexe 2.1)

➢ Port série

On a ensuite tenté de réaliser les entrées/sorties par le port RS232 de la carte-mère SX18 (qui correspond au port /dev/ttyS3 du module Fox) Pour cela il faut tout d'abord ouvrir le port et l'initialiser, en lui indiquant notamment la taille des données et le débit.
Ensuite, on peut réaliser des opérations de lecture et écriture par les fonctions read() et write().

Le programme qui sert d'exemple pour ce type de communication est l'acquisition de données par le port série à partit d'un capteur. Pour se faire, il faut lancer l'acquisition en écrivant la caractère G sur le port série, puis en lisant à intervalles réguliers les données envoyées par le capteur. (cf annexe 2.2)

B) ÉTUDE CARTE SNAP

En cours de projet, on m'a chargée de prendre en main et d'utiliser un module SNAP en plus du module Fox. Pour se faire, on a mis à ma disposition une carte SNAP avec une carte-mère, ainsi que le développeur d'IMSYS (Imsys Developer 5.0).

1. SYSTÈME DE DÉVELOPPEMENT

Imsys Developer est conçu pour fonctionner dans un environnement Windows et n'est pas compatible Linux. Pour pouvoir l'utiliser, j'ai donc installé le développeur sur mon ordinateur personnel puisque les machines utilisées pour le développement des modules Fox étaient toutes sous environnement Linux Debian.

La majeure partie de la documentation disponible concernant le module SNAP est centrée sur les applications Java. Cela est assez logique, étant donnée que le module SNAP est originellement conçu pour le développement en Java. Cependant, les applications que j'étais censée développer devaient être écrites en C. J'ai donc commencé par rechercher de la documentation sur la compatibilité de la SNAP avec le langage C.
Après recherche et lecture du guide de développement (situé dans l'aide du développeur), j'ai fini par déterminer qu'il était possible d'écrire et de compiler des applications en C pour le module SNAP. J'ai également trouver parmi les librairies fournies dans le développeur une librairie nommée macroIO, qui contient essentiellement des fonctions implémentées à partir du langage Assembleur, gérant apparemment les entrées/sorties.

2. COMMUNICATION

Le point suivant de la prise en main de la carte SNAP a été de s'y connecter pour pouvoir y transférer un programme. Pour un premier essai, j'ai choisi d'utiliser le sempiternel programme "hello world", écrit cette fois en Java et fourni en exemple par le développeur.
Suivant la documentation du développeur, le chargement du programme se faisait au moyen d'une connexion par le port parallèle de l'ordinateur. Après re-configuration de la connexion parallèle et installation du driver fourni par le développeur, j'ai tenté une connexion avec la carte SNAP. Sans succès. Après avoir vérifié toutes les erreurs possibles et changé de câble, je suis

arrivée à la conclusion que le guide de développement donnait la procédure à suivre pour un certain type de carte-mère et que la TutorIO que j'utilisais n'en faisait pas partie.
Délaissant le guide de développement, j'ai donc fait des recherches sur la carte-mère et non plus sur le module SNAP. La documentation que j'ai obtenu le site de Taylec (constructeur de la carte-mère TutorIO) propose 2 façons de charger des programmes : soit au moyen d'un système de développement nommé JavaKit, soit par connexion Ethernet. Étant donné que mes applications doivent être écrites en C et que je ne dispose de toute façon pas du JavaKit, la seconde solution m'est apparue comme plus logique. De plus, c'est également la méthode utilisée pour le module Fox. Cependant, les programmes d'exemple fourni par Taylec étaient conçu pour une carte TINI. Bien que le module SNAP soit censément compatible TINI, il n'y a aucune garantie que ce programme puisse fonctionner. Par ailleurs, pour pouvoir effectuer une connexion par Ethernet, il faut disposer de l'adresse IP de la machine que l'on souhaite contacter, ce qui n'est pas mon cas en ce qui concerne le module SNAP.

En conséquence, faute d'avoir pu communiquer avec le module SNAP, je n'ai pas pu ne serait-ce que tester le programme d'essai "hello world".

Conclusion

En conclusion, on peut dire que l'étude n'a pu être achevée par manque de temps et sans doute par défaut d'organisation. Le travail restant à faire dans le cadre de cette étude est d'une part de construire l'application pour le module Fox Board, car les fonctions d'entrées/sorties sont maintenant clairement définies. Pour cela, il faut maintenant assurer la liaison et le transfert des données depuis la carte Fox vers le PC par liaison Ethernet. Une piste qui a été exploitée dans le cadre d'un autre projet est d'utiliser le transport par sockets en protocole connecté ou non. D'autre part, il faudrait poursuivre l'étude du module SNAP, bien que des réserves sur ce sujet soient envisageables. En effet, utiliser le langage C sur un module spécialement conçu pour exploiter du Java n'est peut-être pas l'idéal. Dans le cas du module SNAP, une application Java serait probablement plus efficace. Mais tout d'abord, l'étude devrait être approfondie sur les moyens de communication avec le module.

Sur le plan technique, l'étude m'a permis d'approfondir mes connaissances en langage C et de découvrir "l'envers du décor" sur l'environnement Linux, c'est-à-dire l'installation et l'exploitation d'un système d'environnement différent de Windows, en grande partie sans passer par l'interface graphique.
Sur un plan personnel, ce projet m'a essentiellement permis d'évaluer les points forts et les points faibles de ma façon de m'organiser. Cela me permettra sans doute à l'avenir d'éviter les mêmes erreurs et d'exploiter davantage les bons côtés, ce qui me sera très profitable.

Annexes

Annexe 1.1 : Fonctionnement du SDK Fox – commandes

Pour réaliser la compilation d'un programme, placer la source C et son makefile dans un dossier portant le même nom que l'application dans le répertoire devboard-R_01 directory/apps.

Initialisation des variables :

se placer dans le répertoire devboard-R_01 directory et taper (en tant qu'administrateur)

```
# . init_env
```

Compilation :

Se placer dans le répertoire de l'application.

- Initialisation des librairies :

Pour utiliser les librairies de type glibc, taper

```
# make cris-axis-linux-gnu
```

- Compilation:

Pour réaliser la compilation, taper :

```
# make
```

Transfert de l'exécutable :

Pour réaliser le transfert en scp (ici, l'exécutable de helloworld vers le répertoire flash de la Fox)

```
# scp helloworld root@192.168.0.90:/mnt/flash
```

Lancement de l'exécutable :

Se connecter au module Fox par Telnet :

```
# telnet 192.168.0.90
```

La console demande alors un login et un mot de passe. Les valeurs par défaut sont login : *root* et mot de passe : *pass*

Une fois connecté, lancer l'exécutable:

```
# /mnt/flash/helloworld
```

Annexe 1.2 : Fonctionnement du SDK – programme d'exemple et makefile

Source C

```
#include "stdio.h"

int main(void) {
  printf("Hello world !\n");
  return 0;
}
```

Makefile

Le makefile utilise les variables d'environnement définies à l'initialisation (variables AXIS_TOP_DIR et prefix)

```
AXIS_USABLE_LIBS = UCLIBC GLIBC
include $(AXIS_TOP_DIR)/tools/build/Rules.axis

PROGS     = helloworld

INSTDIR   = $(prefix)/mnt/flash/
INSTMODE  = 0755
INSTOWNER = root
INSTGROUP = root

OBJS = helloworld.o

all: $(PROGS)
$(PROGS): $(OBJS)
        $(CC) $(LDFLAGS) $^ $(LDLIBS) -o $@

install:        $(PROGS)
#       @echo Installing $(PROGS) in $(INSTDIR)
        $(INSTALL) -d $(INSTDIR)
          $(INSTALL) -m $(INSTMODE) -o $(INSTOWNER) -g $(INSTGROUP) $(PROGS)
$(INSTDIR)

clean:
        rm -f $(PROGS) *.o core
```

Annexe 2.1 : Applications d'entrées/sorties – Sortie tout ou rien

Méthode 1 : fonction ioctl()

```
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "sys/ioctl.h"
#include "fcntl.h"
#include "asm/etraxgpio.h"


//programme : envoie 1 puis 0 sur OG25 10 fois (bit sortie uniquement)
int main(void) {
  int fd;
  int i;
  int iomask;

  //ouverture du port
  if ((fd = open("/dev/gpiog", O_RDWR))<0) {
    printf("Open error on /dev/gpiog\n");
    exit(0);
  }

  //masque : selection du bit 25
  iomask=1<<25;

  for (i=0;i<10;i++) {
    printf("Led ON\n");
    ioctl(fd,_IO(ETRAXGPIO_IOCTYPE,IO_SETBITS),iomask);
    sleep(1);

    printf("Led OFF\n");
    ioctl(fd,_IO(ETRAXGPIO_IOCTYPE,IO_CLRBITS),iomask);
    sleep(1);
  }
  close(fd);
  exit(0);
}
```

Méthode 2 : fonctions gpio

```
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "sys/ioctl.h"
#include "fcntl.h"
#include "time.h"
#include "linux/gpio_syscalls.h"

int main(int argc, char **argv) {
    int i;

    //initialisation du bit à 0 (led éteinte)
    gpioclearbits(PORTG, PG25);
    sleep(1);
    //passage du bit à 1 (led allumée)
```

```
    gpiosetbits(PORTG, PG25);

    for (i=0;i<10;i++) {
        printf("Led ON-OFF\n");
      //passage de 0 à 1 ou de 1 à 0
        gpiotogglebit(PORTG, PG25);
        sleep(1);
    }

    return(0);

}
```

Annexe 2.2 : Applications d'entrées/sorties – port série RS232

Source C : fonctions pour l'ouverture et la lecture des données d'un capteur

```
#include <sys/types.h>

#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <getopt.h>

#include <sys/termios.h>
#include <fcntl.h>
#include <stdbool.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include "monserveur.h"

// nombre de bits lus
#define ADXL_READ_LEN 4

// nombre de canaux lus
#define NUM_ADXL_CHANNELS 2

// Length, in chars, of a single data point + channel ID
#define DATUM_LEN 32


// ouverture du port serie RS-232
int adxl_open(const char * portname)
{
    struct termios  term;
    int             comm_fd = -1; //descripteur du port
    const int       baud_rate = B38400;


    /* ouverture du port en lecture écriture */
    comm_fd = open(portname, O_RDWR);

    /* Cas où l'ouverture du port n'est pas possible */
    if(comm_fd <= 0)
    {
      printf("Could not open serial port '%s'\n", portname);
      return(comm_fd);
    }

    // Mise à zéro de la structure
    memset(&term, 0x00, sizeof(term));
```

```
    // recupere les anciens attributs
    tcgetattr(comm_fd, &term);

    // mise à jour des nouvelles données : 8 bits, lecture, 38400 baud
    term.c_cflag = baud_rate | CS8 | CLOCAL | CREAD;
    term.c_iflag = IGNPAR;// No parity
    term.c_oflag = 0;     // Clear these flags
    term.c_lflag = 0;
    term.c_cc[VMIN] = 1;  // Block until 1 char arrives
    term.c_cc[VTIME] = 0; // Inter-character timeout not used

    tcflush(comm_fd, TCIFLUSH);
    tcsetattr(comm_fd, TCSANOW, &term);

    // délai pour mettre à jour les données
    sleep(2);

    printf("initialisation du port serie ok ! \n");

    return(comm_fd);
}

// lecture des données sur le port RS-232
int adxl_read(const int COMM_FD, struct coordonnees *result)
{
    int         itmp;
    char        read_buf[2 * ADXL_READ_LEN] = "";

    // Mise à zéro de la structure
    memset(result, 0x00, sizeof(struct coordonnees));

    // l'acquisition est déclenchée par l'envoi du caractère "G" => ecriture
sur le rs 232 de la fox
    itmp = write(COMM_FD, "G", 1);
    if(itmp <= 0)
    {
      printf("Error %d on serial write \n", itmp);
      return(itmp);
    }

    // lecture des 4 octets - 16 bits (2 octets) pour chaque axe => lecture
sur le rs 232 de la fox
    itmp = read(COMM_FD, read_buf, ADXL_READ_LEN);
    if(itmp < ADXL_READ_LEN)
    {
      printf("Error %d on serial read \n", itmp);
      return(1);
    }

    //  récupération de la partie entière et de la partie décimale => données
du capteur
    result->X = ((read_buf[0] * 256) + read_buf[1])/100.0;
    result->Y = ((read_buf[2] * 256) + read_buf[3])/100.0;
    return(0);
}
```