TP Chaîne d'acquisition - 1

Objectif :
Un nano-ordinateur sous Raspberry OS (Linux) doit recevoir des informations issues d'un système externe (Arduino) via sa liaison série, traiter ses informations pour les afficher ou les enregistrer dans une base de données.

Notions révisées : C++, instance de classe, liaison série, PHP, environnement Linux, outils SSH.

On vous fournit :

- Une platine Arduino avec un logiciel embarqué qui génère des trames GPS, ou qui est capable de recevoir des commandes
- Un nano-ordinateur Raspberry et sa carte SD préchargée.
- Un câble de liaison USB

1 PREPARER LE SYSTEME EXTERNE

Téléversez dans l'Arduino le logiciel fourni, et tester son fonctionnement sur la console.

Fonctionnement :
Mode 1 : (Pin D2 non connectée)
Nous avons inventé un autre protocole « maison », le code SN : Les commandes commencent par SN.
Les commandes possibles sont :

SN	réponse : OK	test de fonctionnement
SNLED0	réponse : OK	Eteint la LED intégrée sur l'Arduino
SNLED1	réponse : OK	Allume la LED intégrée
SNT	réponse : un nombre de secondes	

Mode 2 : (D2 relié à GND)
Le simulateur est en mode GPS (*strap* entre D2 et GND) : il envoi des trames NMEA.
Toutes les trames commencent par le signe « $ » (Norme NMEA). On ne connait pas la longueur de la trame. Le « $ » indique le début de la trame suivante, donc la fin de la précédente…

2 PREPARER LE NANO-ORDINATEUR

- Finalisez l'installation du système Raspberry OS (voir la notice d'installation)

- Connectez-vous en SSH Vous travaillerez en SSH, avec le **client BitVise**.

- Vous utiliserez dans une console le compilateur gcc-g++ et/ou l'interpréteur PHP (selon le groupe) : vérifiez que les 2 sont installés.
- Pour le C++, vous devez copier la bibliothèque sRS232 fournie (**via SFTP** de BitVise) dans votre dossier de développement.

3 TESTEZ LA COMMUNICATION ENTRE LES 2 SYSTEMES

1. Reliez le Raspberry et l'Arduino par la liaison USB.
2. Repérez le port série USB (/dev/tty????????) généré par Linux :
   Commandes à utiliser :
   Liste des périphériques USB : `lsusb`
   Derniers messages système : `dmesg`
3. Vérifiez que vous êtes capables de recevoir des données du simulateur (4800 bauds, 8bits, sans parité) :
   Configurer le port série pour le fonctionnement avec Arduino, grâce à la commande STTY :

```
stty 4800 -echo -opost crtscts –hupcl < /dev/tty???
```

   Explications : On précise la vitesse de transmission (4800 bauds), la suppression de l'écho (-echo) et des fins de lignes Windows (-opost), et la gestion des signaux de contrôle pour éviter que l'Arduino relance son programme à chaque connexion (crtscts et –hupcl).

   Utilisez 2 consoles Linux :
   1 pour recevoir des caractères : `cat < /dev/tty???`
   et 1 pour envoyer une commande : `echo "SN\n" > /dev/tty???`

4 ECRIRE LE PROGRAMME

4.1 En c++ :

1. Utilisez la bibliothèque *sRS232* fournie pour créer un programme qui affiche en boucle à l'écran les trames GPGGA du simulateur.
   Le fichier d'en-tête vous donnera la liste des méthodes de la classe, leurs arguments et leur retour.
   Piste de travail : stockez la trame (repérée par le « $ ») dans une variable C++ de type `std ::string`. Ensuite, testez le 3e caractère de la trame. S'il est égal à « G », c'est une trame GPGGA. Afficher la trame.

   Rappel pour la compilation multi-modules (sans le *makefile*) :

```
g++   sRs232.cpp   mon_main.cpp   –o main
```

   Exécuter le programme :

```
./main
```

2. (IR) Ecrire le code qui va extraire de ces trames la latitude, la longitude, l'altitude, et le nombre de satellites. Vous devrez utiliser les caractéristiques de l'objet c++ std ::*string*, et notamment les méthodes **find** et **substr**.

3. Ecrire un programme qui allume et éteint la LED du l'Arduino toutes les 2 secondes.

4.2 En PHP

L'interpréteur PHP peut exécuter du code PHP sans passer par le serveur Web.
Pour tester un code : **`php ./mon_code.php`**
Le résultat des commandes « echo » apparaitront sur la console.

La classe sRs232 n'étant pas développée en PHP, nous utiliserons les fonctions pures du PHP :
PHP sait accéder aux fichiers ou au port série avec la fonction ***fopen***.
Exemple pour l'envoi de texte :

```
// Ouverture en écriture (r : lexture, r+ : lecture/écriture
$port = fopen("/dev/ttyAMA0","w");
// et configuration du port série en utilisant une commande shell
system("stty 4800 -echo -opost crtscts -hupcl < /dev/ttyAMA0");

$message = "texte à envoyer";
fwrite ($port, $message);
fclose($port);
```

Exemple pour la réception de texte :

```
// Ouverture et configuration du port série
$port = fopen("/dev/ttyAMA0","r");
system("stty 4800 -echo -opost crtscts -hupcl < /dev/ttyAMA0");

// Lecture d'une chaine se terminant par « Fin de ligne »
$car = "";
$trame = "";
while ($car != "\n") // Tant qu'on a pas la fin de ligne
{
      $car = fread ($port, 1);
      $trame .= $car;
}
fclose($port);
echo $trame ;
```

TRAVAIL :

1. Ecrire un programme qui allume et éteint la LED de l'Arduino toutes les 2 secondes.
2. Créer un programme qui affiche à l'écran 1 trame GPGGA du simulateur.
   Piste de travail **:** créez une fonction séparée dont le rôle est de lire la liaison série caractère par caractère jusqu'au caractère « fin de ligne » (code « \n »). Voir l'exemple ci-dessus.
   A la fin du traitement la variable est renvoyée par *return*.
   Le programme principal utilisera cette fonction pour lire les trames.
   Utilisez ensuite la fonction php « explode » pour découper la trame en fonction de la virgule et tester s'il s'agit d'une GPGGA.
3. **(IR)** Modifiez le programme pour qu'il stocke dans des variables distinctes la latitude, la longitude et l'altitude. Faites un « echo » du résultat.
4. **(IR)** Mettez ensuite ces valeurs dans un tableau et convertissez-les au format JSON avec l'instruction « json_encode ». Faites un « echo » du résultat.