Aurore Troubat - Dimitri Bevis
Paul Cartieri - Loïc Vaille

21 Juin 2017

Rapport
Programmation temps réel avec

Groupe Spectrum

Introduction

Pour ce travail, il a été question de récupérer et transmettre en temps réel des données environnementales. Pour cela, nous devions utiliser une carte raspberry et nous servir de Xenomai comme surcouche au kernel Linux. Pour décrire ce que nous avons réalisé, nous avons choisi de décliner ce rapport en deux parties avant de conclure.

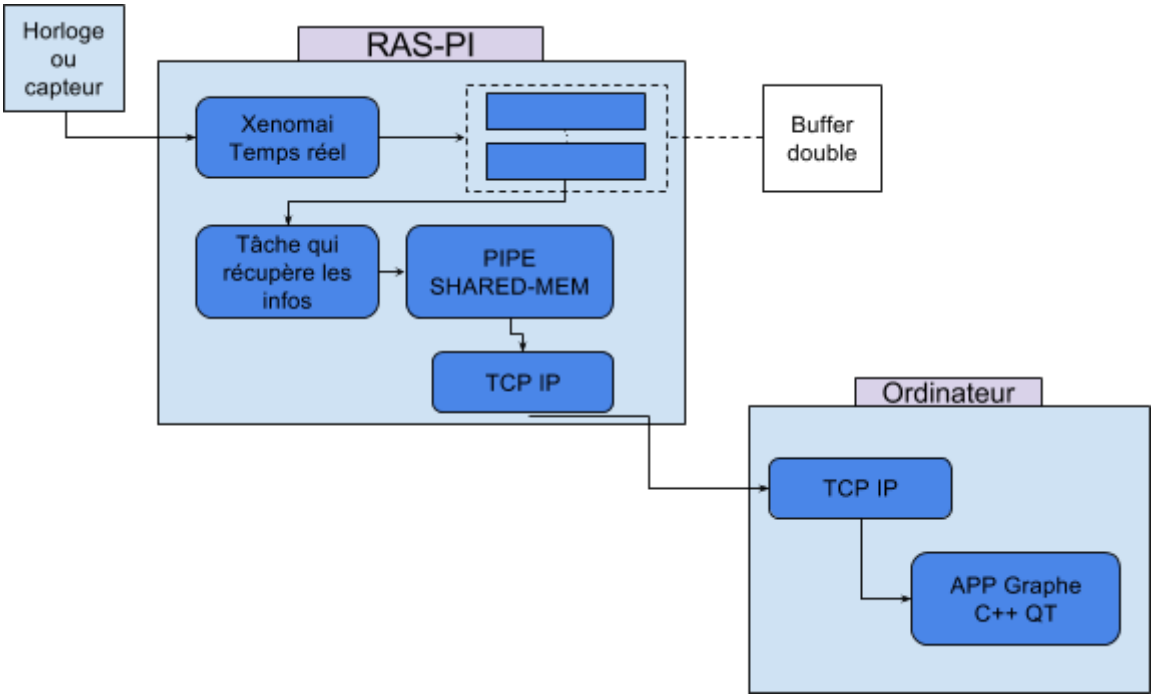
La première présentera les architectures du projet: celle que nous avions prévue initialement et celle que nous avons réalisé en expliquant chaque fois nos choix.

La seconde développera les différentes étapes de réalisation de ce projet, de la partie programmation de la carte à la partie client en passant par la mise en place du temps réel. Nous terminerons par une conclusion où nous expliquerons ce que ce dernier nous a apporté en termes de connaissances.

I - Architecture

A - Architecture initiale

- Plan



- Explications

Notre architecture initiale comme illustrée ci dessus se compose de 3 éléments :

- L'horloge ou capteur
- Le Raspberry Pi
- L' ordinateur

L'horloge ou capteur va permettre la mesure de phénomènes physiques et les transformer en signaux numériques exploitables par le Raspberry Pi. Son but est donc la conversion du phénomène que l'on souhaite mesurer pour permettre le traitement de celui-ci.

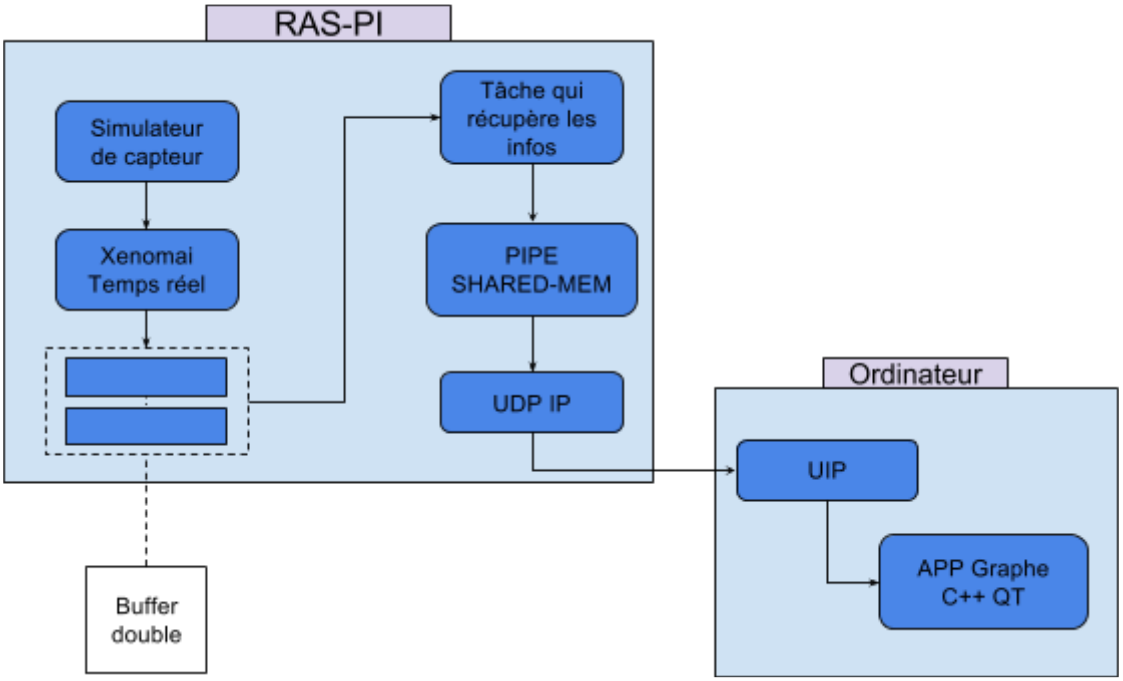
Le Raspberry Pi est un petit calculateur possédant les connecteurs pouvant accueillir l'horloge/capteur. Dans cette configuration le RasPi permet l'exploitation de données en temps réel, la couche Xénomai permet son utilisation "instantanée", le double buffer permet l'utilisation de la mémoire en lecture et en écriture simultanée
Une tâche permet de récupérer les données stockées dans le buffer double, et de les

envoyer dans le Pipe Shared-memory. Le shared-memory sert de tampon le temps que le serveur UDP-IP les envoie.

L'ordinateur équipé d'un serveur UDP-IP va récupérer les données du raspberry et à l'aide de l'application graphe C++ QT va permettre un affichage de ces données sur le moniteur sous forme de graphe.

B - Architecture finale

- Plan



- Explications

Notre architecture initiale comme illustrée ci dessus se compose de 2 éléments :

- Le Raspberry Pi
- l'ordinateur

L'horloge ou capteur va permettre la mesure de phénomène physique et les transformer en signaux numérique exploitable par le Raspberry Pi. Son but est donc la conversion du phénomène que l'on souhaite mesurer pour permettre le traitement de celui-ci.

Le Raspberry Pi est un petit calculateur. Nous lui avons intégré un simulateur de capteur de température.qui va jouer le rôle du capteur réel que nous avions pressenti dans

notre architecture initiale.

Dans cette configuration le RasPi permet l'exploitation de données en temps réel, la couche Xenomai permet son utilisation "instantanée", le double buffer permet l'utilisation de la mémoire en lecture et en écriture simultanée
Une tâche permet de récupérer les données stockées dans le buffer double, et de les envoyer dans le Pipe Shared-memory. Le shared-memory sert de tampon le temps que le serveur UDP-IP les envoyes.

L'ordinateur équipé d'un serveur UDP-IP va récupérer les données du raspberry et à l'aide de l'application graphe C++ QT va permettre un affichage de ces données sur le moniteur sous forme de graphe.

II - Réalisation

A - Partie Raspberry Pi

Installation de Xenomai

Nous sommes partis d'un système Raspbian Jessie installé sur une carte Raspberry 3. Avant d'installer Xenomai à proprement parlé, il a fallu installer la commande rpi-source. Nous avons donc récupéré la dernière version avec un `wget` puis installé la librairie `libncurses5-dev`.

A partir de là, nous avons pu récupérer les paquets de Xenomai 3.0.3 et lancer l'installation via un `make install`. Nous avons ajouté les variables d'environnements nécessaires dans les fichiers `/etc/profile` et `/root/.bashrc`.

La commande `latency` nous a permis de valider sa bonne installation :

```
root@raspberrypi-loic:~# latency -p 100 -T 25
== Sampling period: 100 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 100 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      6.617|      8.408|     30.472|       0|     0|      6.617|     30.472
RTD|      6.572|      8.413|     34.206|       0|     0|      6.572|     34.206
RTD|      6.627|      8.399|     26.300|       0|     0|      6.572|     34.206
RTD|      6.598|     15.922|    187.432|       0|     0|      6.572|    187.432
RTD|     13.389|     16.946|     52.349|       0|     0|      6.572|    187.432
RTD|     13.272|     16.869|     69.312|       0|     0|      6.572|    187.432
RTD|     13.279|     16.914|     52.740|       0|     0|      6.572|    187.432
RTD|     13.384|     16.928|     51.614|       0|     0|      6.572|    187.432
RTD|     13.358|     16.686|     50.964|       0|     0|      6.572|    187.432
RTD|     13.450|     16.676|     53.816|       0|     0|      6.572|    187.432
RTD|     13.329|     16.616|     55.244|       0|     0|      6.572|    187.432
RTD|     13.268|     16.692|     50.874|       0|     0|      6.572|    187.432
RTD|     13.359|     16.629|     50.548|       0|     0|      6.572|    187.432
RTD|     13.347|     16.696|    185.690|       1|     0|      6.572|    187.432
RTD|     13.490|     16.668|     54.950|       1|     0|      6.572|    187.432
RTD|     13.370|     16.633|     53.844|       1|     0|      6.572|    187.432
RTD|     13.499|     16.722|     58.396|       1|     0|      6.572|    187.432
RTD|     13.348|     16.644|     58.279|       1|     0|      6.572|    187.432
RTD|     13.231|     16.694|     50.889|       1|     0|      6.572|    187.432
RTD|     13.375|     16.638|     52.543|       1|     0|      6.572|    187.432
RTD|     13.362|     16.665|     50.708|       1|     0|      6.572|    187.432
RTT|  00:00:22  (periodic user-mode task, 100 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|     13.350|     16.649|     57.811|       1|     0|      6.572|    187.432
RTD|     13.389|     16.677|     80.718|       1|     0|      6.572|    187.432
RTD|     13.455|     16.675|     64.765|       1|     0|      6.572|    187.432
---|-----------|-----------|-----------|--------|------|-----------------------
RTS|      6.572|     15.644|    187.432|       1|     0|    00:00:25/00:00:25
root@raspberrypi-loic:~#
```

Emulation d'un capteur de température

Nous avons simulé un capteur de température via un random:

`rel.temperature = rand()%40`.

B - Programme de temp réel

Création des tâches temps-réel

Dans le programme nous créons deux buffers grâce à deux threads Posix et tâches Xenomai. Nous les paramétrons grâce au `pthread_attr` afin de définir des prioritées et des vitesses d'exécution.
Les tâches exécutent deux fonctions : une pour lire la valeur sondé, une autre pour envoyer la valeur sur un socket UDP-IP.

Les deux fonctions ont un mutex partagé qui permet de lire des valeurs entre les threads.

La première fonction de lecture écrit dans le mutex la date et la température (valeur aléatoire). Pour pouvoir écrire en sécurité, nous bloquons l'accès au mutex pour le relâcher une fois les valeurs mises à jour.

La seconde fonction va lire le mutex et envoyer la valeur de la température par un socket UDP-IP. Pareil nous bloquons l'accès au mutex pour pas que la valeur change pendant l'envoi, nous la relançons une fois le message envoyé.

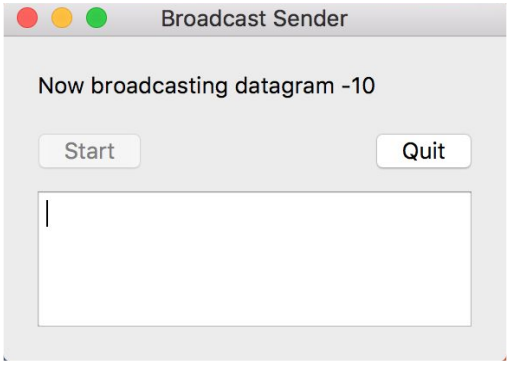
Tout ceci tourne tant que l'utilisateur n'a pas coupé l'exécution du programme. Les relevés se font environs toutes les 500ms.

C - Partie Ordinateur

QT Créator

Serveur UDP:

Nous avons créé un serveur UDP de test, ce serveur envoie des valeurs situées entre 50 et -50. Son but est uniquement de nous permettre de tester que le client reçoit bien des informations en local. Il utilise le nom de la machine en local et le port 45454.

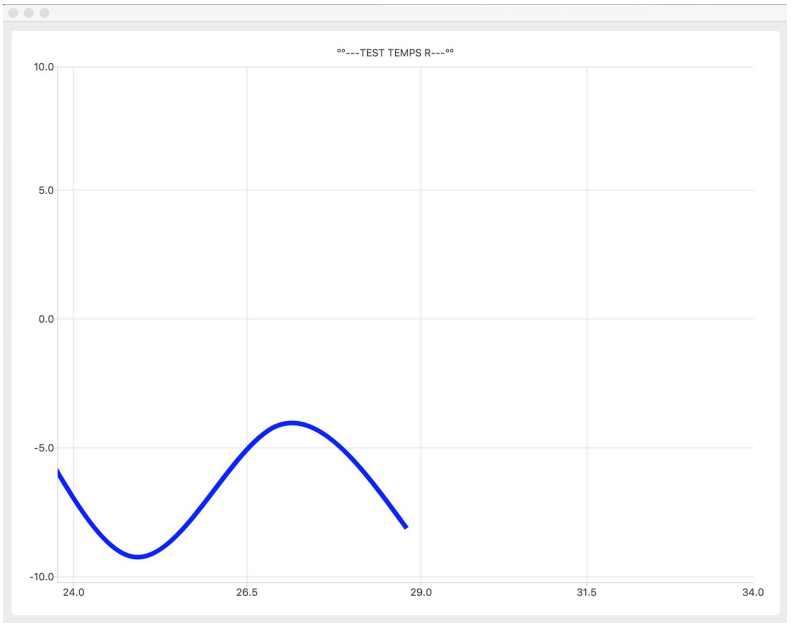


Client UDP:

Le client UDP à été intégré à un graphique de type “chart”. Il récupère les informations envoyées par le serveur et les affiche avec une courbe. Pour se faire, le programme calcule les coordonnées en abscisse “X” et ordonnée “Y” pour afficher un point sur le graphique.
Au fur et à mesure du temps et qu'on reçoit des informations, la courbe variera entre 50 et -50.
Il écoute sur le port 45454.

Bibliothèques utilisées :

```
#include <QtCharts/QAbstractAxis>
#include <QtCharts/QSplineSeries>
#include <QtCharts/QValueAxis>
#include <QtCore/QTime>
#include <QtCore/QDebug>
#include <QtWidgets>
#include <QtNetwork>
#include <QString>
```



Conclusion

Ce projet a été extrêmement formateur pour nous quatre. En effet, il nous a permis de monter en compétences dans différents axes.

Tout d'abord, nous avons pu évoluer dans un environnement intégralement open-source via des systèmes Raspbian ou Ubuntu.

Aussi, nous avons mis en pratique nos connaissances acquises en réseau via les différentes connexions entre les blocs applicatifs du projet.

Enfin, nous avons amélioré notre connaissance du langage C et de son écosystème avec notamment l'interfaçage Qt.

Code source

Lien Github : https://github.com/Hydrog3n/TempReel