Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

2021/2022

Informatique Embarquée

Rapport 02

Travail Pratique 2 : Station météo, écran LCD et *threads*

Ecole	Haute école d'ingénierie et d'architecture de Fribourg (HEIA-FR)
Branche	Informatique embarquée
Etudiants	Barras Simon, Terreaux Nicolas
Groupe	A1
Classe	ISC-IL-2d
Professeur	Bovet Patrick
GitLab	https://gitlab.forge.hefr.ch/embsys/2021-2022/classe-patrick-bovey/gra1/tp02
Version/Date	V1 du 09.03.2022

1 Introduction

Dans ce TP, nous allons intégrer un nouveau capteur environnemental (BME280) qui permet la mesure de température, d'humidité relative et de pression atmosphérique. Il faudra dans un premier temps récupérer ces données puis les afficher sur l'écran LCD de la cible. La dernière étape sera la transformation de l'application en programme multi-tâches.

2 Résumés

Ce chapitre comprend le résumé des différentes notions apprises durant le TP classé de la manière suivante : non-acquis, à exercer ou parfaitement acquis.

2.1 Non Acquis

Nous ne pensons pas avoir de choses non-acquis.

2.2 À exercer

Acquisitions des données environnementales

Nous avons réalisé cette partie étape par étape. Dans un premier temps, nous faisions ça de manière séquentielle puis nous avons rajouté la partie multi-thread étape par étape. Nous manquons d'entraînement pour faire tout ça du premier coup.

Threads (multi-tâches)

Le concept ainsi que l'implémentation est clair mais il nous faudrait encore de la pratique pour être efficace et être sûr de notre code en c++. De plus, nous découvrons la notion de « Mail » qui est propre à « Embed ». La mise en place du mode d'acquisition urgent a fonctionné rapidement en utilisant une priorité lors de la création du thread.

Display LCD

Nous avions déjà utilisé le display dans le cours d'architecture des ordinateurs donc nous savions déjà comment il fonctionne. Cependant, nous ne sommes pas encore à 100% à l'aise. Nous perdons un peu de temps pour l'utiliser car nous faisons pas mal d'essais-erreurs.

2.3 Parfaitement acquis

CI/CD :

Le CI/CD est une chose que nous exerçons depuis le début de l'année dans tous les cours qui s'y prêtent. Cependant, nous apprécions toujours et le faire et nous comprenons sont importance dans l'industrie. Nous sommes capables de comprendre un CI/CD existant mais aussi d'en créer un nous-même afin de pouvoir personnaliser les tests au cas par cas.

Tests unitaires :

Capteur : Pour les tests unitaires, nous avons tout d'abord tester le capteur environnemental en récupérant une premières fois les différentes données (température, pression atmosphérique et humidité). Chaque donnée est analysée et contrôlée. La température doit être entre 10°C et 35°C, la pression atmosphérique doit être entre 900mbar et 1000mbar et l'humidité entre 10% et 60%.

Si tout est ok, on refait une mesure (5 secondes plus tard) et on fait la différence des mesures au carré (au cas où il y a des valeurs négatives) et on check si les différences sont inférieures à 10.

(Nous avons testé les négatif en utilisant une bombe de gaz comprimé.)

Display : Ensuite, on affiche différentes valeurs sur le display pour voir si l'affichage et le clear sont correct. Les valeurs ne sont pas prises par hasard mais nous testons des extrêmes pour voir si tout est gérer.

Pour la température, nous testons les négatifs ainsi que les positifs. Pour la pression, des valeurs inférieures à 0 et pour l'humidité des valeurs en dessous de 0% et en dessus de 100% (ce qui n'est pas possible) donc nous affichons respectivement la limite. Nous avons bien entendu aussi tester des valeurs correctes.

Bouton : Nous ne pouvons malheureusement pas automatiser les tests du bouton car ce n'est pas vraiment possible avec nos moyens de tester quelque chose qui a besoin d'une interaction physique et en plus les cibles sont à distances. C'est pourquoi nous n'avons pas fait de tests unitaires pour le bouton.

MBed statistics : Pour démontrer les proportions de temps que le CPU passe en mode actif (Up), inactif (Idle), Sleep et Deep Sleep, nous avons (comme il a été demandé) utiliser MBed Statistics.

RESULTATS :

```
Testing...
If you don't see any output for the first 10 secs, please reset board (press reset button)

Temperature 1 =  24.150000
Pressure 1 =  932.114136
Humidity 1 =  21.385742
test\test_compile.cpp:69:thermo_test:INFO: Temperature is greater than 10 degrees, pressure is greater than 900mbar and humidity is greater than 10%
test\test_compile.cpp:75:thermo_test:INFO: Temperature is lower than 10 degrees, pressure lower than 1000mbar and humidity lower than 60%
Temperature 2 =  24.110001
Pressure 2 =  932.126404
Humidity 2 =  21.686523
Difference temp =  0.039999
Difference pres =  -0.012268
Difference hum =  -0.300781
test\test_compile.cpp:110:thermo_test:INFO: The difference between the first and second measurement of all arguments are less than 3 (degrees, mbar and %)
test\test_compile.cpp:175:thermo_test   [PASSED]
test\test_compile.cpp:176:display_test  [PASSED]
Tite(us): Up: 28031372   Idle: 27181884   Sleep: 27181884   DeepSleep: 0
Idle: 79% Usage: 21%s
t\test_compile.cpp:177:time_stat_test   [PASSED]
-----------------------
3 Tests 0 Failures 0 Ignored
======================================================================== [PASSED] Took 48.66 seconds ========================================

Test    Environment    Status    Duration
------  -------------  --------  ------------
*       DISCO_F        PASSED    00:00:48.663
========================================================================= 1 succeeded in 00:00:48.663 =======================================

Terminal will be reused by tasks, press any key to close it.
```

3 Points importants

Ce chapitre comprend tous les points qui nous semblent importants et que nous devons retenir.

- Acquisitions des données environnementales (et différents modes)
- Threads
- Mail
- Display LCD
- Capteur environnemental
- CI/CD
- Tests unitaires

4 Conclusion

Nous avons apprécié effectuer ce travail. Travailler avec quelque chose de physique est assez satisfaisant. Nous avons été efficace pour la réalisation et nous avons bien compris les points importants abordés durant l'implémentation. C'est la première fois que nous utilisions du multi-tâches dans un système embarqué.

Nous sommes satisfaits de notre travail et nous sommes prêts à attaquer les prochains travaux. Nous n'avons (au jour du rendu) pas encore reçu de feedback sur le TP1… Donc nous n'avons pas pu corriger les erreurs (s'il y en a) pour ce TP.

Nous avons passé chacun environ **6h** en dehors des cours.

5 Code source

Nous voulons expliquer le fonctionnement du bouton ISR car la gestion des interruptions nous paraît important.

```
ButtonISR::ButtonISR(PinName pin, SensorDataServer* sensor)
    : button_(pin), sensor_(sensor), control_led_(LED2, kLedOff)
{
    // Produc an interruption when the button is fall
    button_.fall(callback(this, &ButtonISR::fall_handler));
}

void ButtonISR::start_handling()
{
    // Launch the thread that look at the button
    thread_.start(callback(this, &ButtonISR::dispatcher));
}

void ButtonISR::dispatcher()
{
    // Put queue's thread in high priority
    osThreadSetPriority(osThreadGetId(), osPriorityHigh);
    e_queue_.dispatch_forever();
}

void ButtonISR::fall_handler_deferred(void)
{
    // this is executed in thread context
    tr_info("Button pressed");
    sensor_->pollingData();  // Fetch data when there is a event in
the queue
    control_led_ = kLedOff;
}

void ButtonISR::fall_handler(void)
{
    // this is executed in ISR context
```

```
    control_led_ = kLedOn;
    // Add instruction to the queue
    e_queue_.call(callback(this, &ButtonISR::fall_handler_deferred));
}
```

Pour mieux comprendre ce qui se passe lorsque lors que nous pressons un bouton, nous allons vous raconter l'histoire de « Eve » le petit événement.

Il était une fois un humain qui a pressé un bouton sur une cible à la HEIA-FR. Comme indiqué dans le constructeur, cette action entraine la naissance d'Eve dans la méthode « fall_handler ». Dans un premier temps, Eve est placé dans une couveuse (e_queue_.call) jusqu'à que ses parents puissent s'occuper de lui. Remontant un peu en arrière et regardons la vie de ses parents. Ils ont débuté leur existence grâce à la méthode « Start_dispatching ». Cette dernière leur a donnée la mission de s'occuper des events comme Eve jusqu'à la fin des temps (e_queue_.dispatch_forever). Comme leur mission est importante., ils peuvent rapidement accéder aux couveuses (osThreadSetPriority high). Maintenant que nous avons expliqué le rôle des parents, nous pouvons revenir à Eve. Lorsque les parents vont récupérer Eve (fall_handler_deferred), ils vont faire plein d'activités avec leur petit event comme regarder la météo (sensor_->pollingData). Malheureusement, une fois que les activités sont finies, Eve disparaît à jamais.

Même si cette histoire est très enfantine, nous trouvons qu'elle résume très bien la situation.