1. Préambule

Au long de ce module, vous allez utiliser 3 nano-ordinateurs (voire 4 si vous le souhaitez) (RaspberryPi 1B, RaspberryPi 3B, ESP32 TTGO-Lora32-Oled et en option Arduino nano), quelques périphériques (LED, bouton, photorésistance, PIR, accéléromètre (en option), etc.) et mettre en oeuvre plusieurs "briques technologiques" intervenant dans les systèmes d'objets communicants (HTTP, MQTT, sql, i2c, Arduino, etc.).

Vous allez travailler en binôme. Le choix de votre binôme se fait à la première séance et il est **définitif** (svp). Dans ce TP, vous allez écrire une application pour RaspberryPi 1B afin de contrôler deux LEDs et un bouton un bouton poussoir. Cette application aura trois threads POSIX, 1 par LED et 1 pour le bouton-poussoir. Ce que vous allez devoir faire, c'est :

- Compilation croisée via un compilateur déporté.
- Connexion à un système distant par ssh / scp.
- Manipulation directe des GPIO d'une RaspberryPi en mode utilisateur.
- Écriture d'application multi-threadées simple.

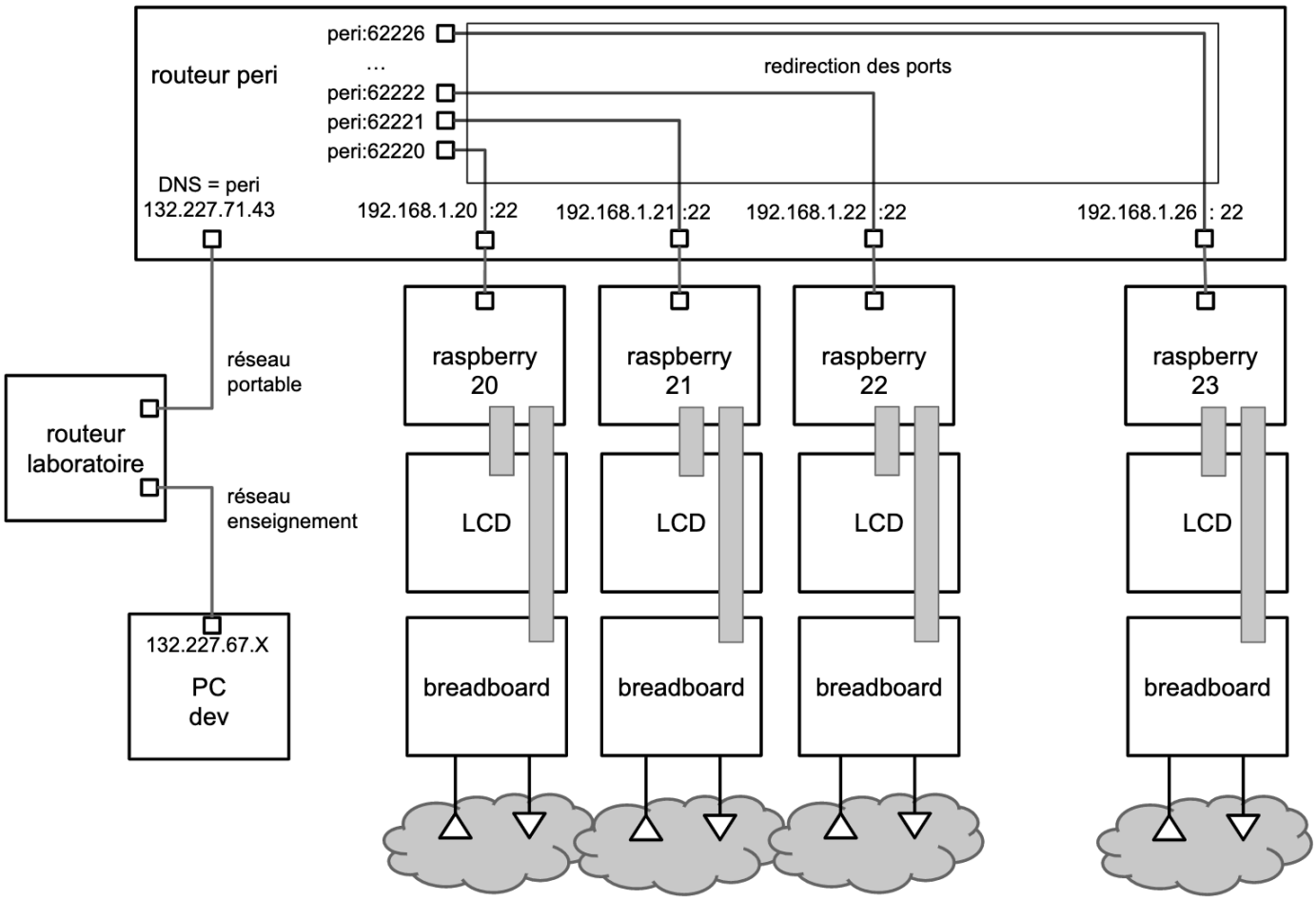
Vous écrirez un **compte-rendu au format markdown** (CR), dans lequel vous répondrez aux quelques questions du TME, vous mettrez les étapes de réalisation du TP, ainsi vous conserverez une trace et vous ajouterez tout ou partie de vos codes commentés (succinctement, mais proprement). Les comptes-rendus de TP doivent être suffisamment explicites pour que vous puissiez les réutiliser plus tard. Par exemple, vous allez devoir lire des documents externes et vous devrez référencer ces documents (mettre les liens) pour les retrouver. En fonction de vos connaissances de départ, vous pourrez être plus ou moins explicite sur tel ou tel point.

- Le document doit impérativement être nommé `ioc23_X_nom1_nom2.md` (où X est le numéro du TP: 1 pour le TP1, 2 pour le TP2, etc.)
- vos noms doivent aussi être présents dans le document.
- Pour l'édition, je vous conseille de vous créer un compte gratuit sur HackMD, vous pourrez faire une édition collaborative avec votre binôme.
- Vous devez déposer le fichier markdown du compte-rendu sur Moodle.

Si vous préférez un autre format que markdown (comme LaTeX ou Word), c'est possible, mais dans ce cas vous devez fournir le PDF de vos comptes-rendus. Toutes les autres remarques, sur le nom des fichiers et leut contenu restent valables.

2. Hello World! RaspberryPi

Le schéma ci-dessous représente la plateforme des cartes RaspberryPi 1. Pour exécuter votre programme sur une carte RaspberryPi, vous devez vous connecter en `ssh` sur une des cartes RaspberryPi en passant par le routeur peri. Le routeur peri a deux adresses: `132.227.71.43` du côté laboratoire et `192.168.1.1` de l'autre coté. Le serveur de nom (DNS) du laboratoire a attribué le nom `peri` à l'adresse `132.227.71.??`. Le routeur peri crée un sous-réseau (`192.168.1.x`) où sont connectées les cartes RaspberryPi. Les cartes sont numérotées de `20` à `26`. Le routeur a été configuré pour reconnaître les adresses MAC des cartes et leur attribuer une adresse IP fixe. La carte n°`X` (`X` allant de 20 à 26) reçoit l'adresse IP 192.168.1.`X` (par exemple la carte n°`20` a comme adresse IP sur ce réseau `192.168.1.20`). Pour faire ça, le firmware du routeur a été remplacé par **dd-wrt** qui permet ce type de configuration. Le routeur a été également été configuré pour que tous les paquets entrants sur le port `62200`+`X` de l'adresse `peri` soient routés vers la carte RaspberryPi d'adresse IP `192.168.1.X` port 22. Le port 22 est celui sur lequel écoute le serveur ssh.



Il n'y a qu'un seul compte utilisateur sur les cartes RaspberryPi. Le nom de login est `pi` et le mot de passe est `raspberry` (ne le changez pas). Comme, il n'y a qu'un compte par carte et que vous allez devoir vous le partager, vous devez créer un répertoire à la racine avec **votre nom de votre login** sur le réseau enseignement accolé avec un underscore (_) avec le **nom de login** de votre binôme. Par exemple, deux personnes en binôme, dont les noms de login sont respectivement `almada` et `fomentin` vont créer un répertoire `almada_fomentin` (en minuscule). En principe, vous allez toujours utiliser la même RaspberryPi, mais vous pouvez vous déplacer en cas de problème, cependant veillez à une répartition équitable.

> **Je vous demande de respecter cette convention**
> **afin que je puisse vous identifier facilement.**

Pour se connecter en ssh sur la carte n° `20` avec le login `pi`, il faut utiliser le port 62220 de l'adresses IP `peri` (132.227.71.43) et les paquets sont routés sur l'adresse 192.168.1.20 port 22, il faut taper (remarquez que c'est un `p` minuscule)

```
$ ssh -p 62220 pi@peri
pi@peri's password: raspberry
```

Pour copier un fichier `file.x` depuis votre compte enseignement sur le carte n° `20` il faut taper (remarquez que c'est un `P` majuscule):

```
$ scp -P 62220 file.x pi@peri:almada_fomentin
pi@peri's password: raspberry
```

Il est recommandé de ne pas laisser de fichiers source sur la carte SD d'une RaspberryPi, car celle-ci peut être reformatée en cas de dysfonctionnement.

Questions pour voir si vous avez compris

1. Pourquoi passer par la redirection des ports ? (faites une réponse succincte)
2. Pourquoi faut-il que vos fichiers soit dans un répertoire propre sur une RaspberryPi ?

3. Configuration des clés ssh

Taper le mot de passe à chaque fois que l'on se logue et à chaque copie peut être pénible à la longue. Pour éviter cela, il faut donner à votre RaspberryPi la clé publique du protocole de chiffrement utilisé pour la connexion. Vous allez utiliser les commandes suivantes sur le PC développement :

```
ssh-keygen -t rsa
ssh-copy-id -i $HOME/.ssh/id_rsa.pub -p 622x pi@peri
ssh -p 622x pi@peri
```

- Pour la première commande, vous devez taper 3 fois sur la touche entrée pour choisir les valeurs par défaut.
- Pour la deuxième commande, vous sélectionnez la bonne carte (en remplaçant x par le bon chiffre) et vous tapez le mot de passe `raspberry` (ce sera la dernière fois).
- La troisième est là pour vérifier que vous n'avez plus besoin du mot de passe.
- Notez que vous pouvez utiliser cette technique pour vous passe facilement d'une machine à l'autre, mais vous devez retirer l'option de redirection des ports (-p xxxx).

4. Prise en mains des outils de développement: Hello World!

La première étape consiste à vous familiariser avec les outils de développement. Pour cela, vous allez développer un petit programme de type "Hello World!" qui affiche une phrase sur la sortie standard (c.-à-d. le terminal) grâce à un printf. Pour compiler votre programme, suivez les instructions suivantes :

- Tout d'abord, configurez votre terminal pour utiliser le compilateur croisé. Ajoutez dans votre $HOME/.bashrc (sur le PC de développement) la ligne suivante et ouvrez un nouveau terminal.

```
source /users/enseig/franck/IOC/export_rpi_toolchain.sh
```

- Une fois le terminal configuré, vérifiez que le compilateur est accessible, si cette commande ne retourne rien, la configuration n'a pas fonctionné.

```
which bcm2708hardfp-gcc
```

Votre suite d'outils (toolchain) contient tous les outils nécessaires pour la compilation, l'édition des liens et la manipulation de binaires pour la carte RaspberryPi. Tous ces outils sont préfixés par la même chaîne de caractères : `bcm2708hardfp-`. Il s'agit donc d'un compilateur pour le SoC BCM2708 avec l'option hardfp activée (calcul flottant matériel). Il s'agit bien du SoC de la carte RaspberryPi 1B.

Maintenant, pour compiler un programme C vers un binaire qui puisse s'exécuter sur la carte RaspberryPi, il vous faut écrire un Makefile pour plus de facilité. Pour cela, suivez la syntaxe de base des Makefile:

```
cible: dépendances
	commande
```

Notez bien que l'indentation de la seconde ligne doit OBLIGATOIREMENT être une tabulation et non une suite d'espaces. Vous pourrez donc par exemple, écrire la règle de Makefile suivante:

```
helloworld.x: helloworld.c
	bcm2708hardfp-gcc -o $@ $< -O2 -static
```

Ci-dessous, un `Makefile` un peu plus complexe qui se charge de la copie et qui utilise la règle de compilation implicite. Dans la suite, nous vous fournissons un nouveau Makefile que vous pourrez modifier.

```
CC=bcm2708hardfp-gcc
CFLAGS=-O2 -static
CARD=20
NAME=nom1-nom2
CFLAGS=-W -Wall -Wextra -Wfatal-errors -O2
APP=helloworld

all: $(APP)
	scp -P 622$(CARD) $^ pi@peri:$(NAME)
```

```
clean:
	rm $(APP)
```

Attention aux tabulations

Devant les commandes (scp et rm) vous devez mettre une tabulation. Si vous n'êtes pas familier avec les makefiles, consultez l'article de Wikipedia ou de Developpez.com.

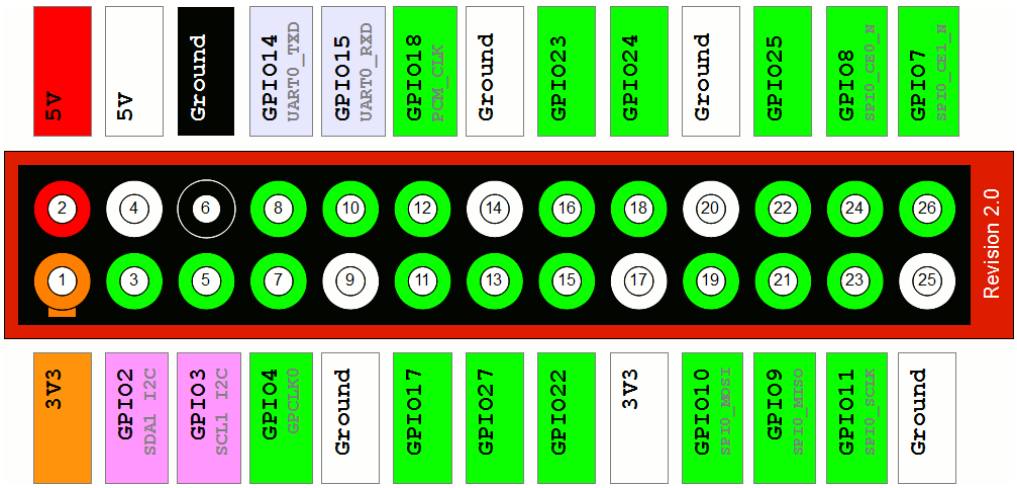
Option -static

L'option "-static" est utilisée par l'éditeur de lien. Elle est importante ici, car la librairie C du compilateur croisé n'est pas tout à fait identique à la librairie C sur la carte RaspberryPi. Ajouter "-static" à la ligne de compilation permet de créer un binaire qui contient en plus les fonctions de la librairie C utilisée par votre programme. Ceci permettra à celui-ci de ne pas essayer d'utiliser des fonctions de la librairie C installée sur la carte qui, sinon, aurait été chargée dynamiquement.

Vous devez donc :

- Créer un répertoire `helloworld` sur le PC de dev et vous y déplacer
- Créer un fichier C `helloworld.c` avec seulement la fonction `main()` qui affiche "Helloworld !" sur stdout.
- Créer un Makefile sur le modèle du dernier exemple en le modifiant.
- Crosscompiler et uploader l'exécutable sur votre RaspberryPi
- Exécuter le programme.

5. Contrôle de GPIO en sortie



Dans cet exercice, on vous propose de manipuler une pin du GPIO en mode "sortie" pour contrôler le clignotement d'une LED à une fréquence donnée.

- Récupérer le répertoire lab1, exécuter la commande suivante dans un terminal et dans le dossier que vous souhaitez pour contenir votre code pour ce premier TP.

```
cp -rp /users/enseig/franck/IOC/lab1 .
```

- Éditez le fichier blink0.c
  ATTENTION: ne changez pas les valeurs de GPIO_LED0, GPIO_LED1 et GPIO_BP, car vous risqueriez d'endommager la carte RaspberryPi.
- Ensuite, compilez-le grâce au Makefile (**Vous devez l'adapter à votre carte et vos noms**) qui vous est fourni.
- Exécuter le avec `sudo` sans argument et changer la fréquence de clignotement en passant un argument.

```
sudo ./blink0
```

Questions sur le code de blink0.c

1. Expliquez pourquoi, il pourrait être dangereux de se tromper de broche pour la configuration des GPIO.
2. A quoi correspond l'adresse `BCM2835_GPIO_BASE` ?
3. Que représente la structure `struct gpio_s` ?

4. Dans quel espace d'adressage est l'adresse `gpio_regs_virt` ?
5. Dans la fonction `gpio_fsel()`, que contient la variable reg ?
6. Dans la fonction `gpio_write()`, pourquoi écrire à deux adresses différentes en fonction de la valeur val ?
7. Dans la fonction `gpio_mmap()`, à quoi correspondent les flags de `open()` ?
8. Dans la fonction `gpio_mmap()`, commentez les arguments de `mmap()`.
9. Que fait la fonction `delay()` ?
10. Pourquoi doit-on utiliser `sudo` ?

6. Contrôle de plusieurs GPIO en mode "sortie"

Vous allez maintenant faire clignoter deux LEDs à des fréquences différentes et paramétrables sur la ligne de commande. Pour tous les exercices ci-après, vous devez changer votre Makefile en ajoutant les programmes à compiler. Vous devez utiliser des Posix threads, si vous ne savez pas ce que c'est, c'est le moment d'apprendre, vous trouverez des tutoriels sur le net, par exemple developpez.com/pthread

- Commencez par faire une copie du fichier `Blink0.c` en `blink0_pt.c`. Vous allez créer un thread Posix dans `blink0_pt.c` pour faire clignoter une led, c'est donc le même comportement que blink0 avec un pthread pour la partie `// Blink led ...` (à la fin de `main()`).
- Faites ensuite une une copie du fichier `Blink0_pt.c` en `blink01_pt.c` et faites clignoter les deux LEDs en utilisant deux pthreads.

7. Lecture de la valeur d'une entrée GPIO

Maintenant que vous maîtrisez le contrôle d'un GPIO en sortie, passons au mode "entrée".

- Écrivez un programme `read_bp.c` qui configure la GPIO 18 en entrée (là où est connecté le bouton-poussoir) et qui affiche la valeur de ce GPIO dans une boucle infinie.
- On veut maintenant détecter l'appui sur le bouton-poussoir. Pour cela, reprenez le programme `blink01_pt.c` et faites une copie en `blink01_bp_pt.c`. Dans un pthread, vous allez lire la valeur du bouton périodiquement toutes les 20ms. L'appui ou le relâchement est présent quand deux valeurs lues successivement sont différentes. Quand le bouton n'est pas enfoncé, la valeur de la pin est à 1.
- Commencer par tester que vous savez détecter l'appui sans gérer les LEDs.
- Ensuite, modifier le programme pour faire fonctionner la led0 en télérupteur (un appui changer l'état de la LED) alors que la led1 continue de clignoter périodiquement.

Principe d'utilisation du changement d'état du bouton

Deux variables globales pour le passage de l'information entre la tâche (le thread) qui gère le bouton et détecte les événements d'appui et la tâche qui attend ces événements. `BP_ON` et `BP_OFF` sont un peu des *lignes d'interruption* qui sont levées (mise à 1) par la tâche qui gère le bouton et baissées (acquittées, mise à 0) par la (ou les tâches) qui attendent après qu'elles aient vu le passage à 1.

```
BP_ON = 0   // mis à 1 si le bouton a été appuyé, mis à 0 quand la tâche qui attend l'appui a vu l'appui
BP_OFF = 0  // mis à 1 si le bouton a été relâché, mis à 0 quand la tâche qui attend le relâchement a vu le relâchement
```

Pseudo-code de la tâche de détection du changement d'état du bouton

```
val_prec = 1
val_nouv = 1
faire toujours
   attendre 20ms
   val_nouv <- valeur du BP
   si (val_prec != val_nouv) alors    // changement d'état
      si (val_nouv == 0) alors        // appui détecté
         BP_ON <- 1
      sinon                           // relâchement détecté
         BP_OFF <- 1
      fsi
      val_prec = val_nouv
   fsi
finfaire
```

Pseudo-code d'une tâche qui attendrait à la fois l'appui et le relâchement du bouton. Notez que dans le cas du télérupteur, seul un des deux changements d'état est utilisé.

```
si BP_ON alors
   BP_ON <- 0                    // l'appui est un événement ponctuel
   comportement quand un appui est detecte
fsi
si BP_OFF alors
   BP_OFF <- 0                   // Le relâchement est un événement ponctuel
   comportement quand un relâchement est detecte
fsi
```

8. Pour les plus motivés

Les leds et le bouton poussoir, bien que très simples, permettent de faire des expérimentations intéressantes. Par exemple, vous pouvez faire un programme qui fait clignoter la led1 avec une fréquence fixe alors que la led2 clignote à une fréquence dépendante de l'appui du bouton. Un appui et la led2 est éteinte, un nouvel appui et la led2 clignote à la fréquence de la led1 en phase, un nouvel appui et la led2 clignote en opposition de phase, un dernier appui et la led2 est à nouveau éteinte. Ce n'est pas trop difficile. On peut imaginer des scénarios beaucoup plus complexes :-)

Compte rendu

Vous devez rendre un compte-rendu ioc23_X_nom1_nom2.zip avec

- ioc23_X_nom1_nom2.md contenant la description des expériences et les réponses aux questions
- 2 répertoires avec vos sources *correctement* commentées (juste le nécessaire).
  - `helloword` avec `helloworld.c` et `Makefile` `
  - `lab1` avec `Makefile`, `blink0.c`, `blink0_pt.c`, `blink01_pt.c`, `read_bp.c`, et `blink01_bp_pt.c`.
  - `lab1+` avec d'autres programmes si vous les avez faits :-)

Last modified on Jan 25, 2024, 4:52:31 PM

Attachments (1)