

# Compte rendu

Plateformes de développement de l'IOT : Cloud et OS, Communication logicielle

Paul Lamy  
Matthieu Seigeot  
IR 2019

# Sommaire

1. TP 1
  1. Configuration de la Raspberry Pi
  2. Configuration de la machine virtuelle
2. TP 3
  1. Compilation du noyau
  2. Test sur émulateur
  3. Test sur Raspberry Pi
3. Projet
  1. Création du montage
  2. Configuration de l'Arduino
  3. Configuration de la Raspberry Pi
  4. Création de l'application Android

# TP 1

## 1. Configuration de la Raspberry Pi

Nous avons commencé par récupérer l'image du système Raspbian puis l'avons placée sur une carte microSD. Afin de pouvoir communiquer avec la Raspberry Pi en ssh, on ajoute un fichier portant le nom « ssh ». De cette manière, le système activera ce type de connexion. On branche ensuite la Raspberry Pi en Ethernet sur notre ordinateur, on la branche sur un port USB et on configure notre ordinateur pour faire office de serveur DHCP pour elle grâce au logiciel Tftpd64.

De cette manière, nous pouvons accéder à notre Raspberry pi en ssh grâce à Putty en rentrant son adresse ip. Une interface en ligne de commande apparaît et nous permet de saisir l'identifiant « pi » et le mot de passe « raspberry » afin de valider la connexion. Une fois cela effectué, nous avons accès au contenu de celle-ci et pouvons effectuer des commandes pour mettre à jour la listes des paquets par exemple.

## 2. Configuration de la machine virtuelle

On crée sur l'ordinateur une machine virtuelle contenant Ubuntu 14 afin de pouvoir écrire du code, le cross-compiler puis envoyer l'application sur la raspberry pi où elle y sera lancée.

Une fois la machine virtuelle opérationnelle, on installe les ressources nécessaires pour pouvoir cross-compiler :

```
wget https://s3.amazonaws.com/RTI/Community/ports/toolchains/raspbian-toolchain-gcc-4.7.2-linux64.tar.gz
tar xvf raspbian-toolchain-gcc-4.7.2-linux64.tar.gz
export PATH=$(pwd)/raspbian-toolchain-gcc-4.7.2-linux64/bin:$PATH
echo $PATH
```

On peut maintenant cross-compiler notre programme codé en C/C++ avec cette commande :

```
arm-linux-gnueabi-gcc test.c -o test
```

Une fois celle-ci effectuée, on obtient l'application compilée pour fonctionner sur la raspberry pi. Il faut donc maintenant l'envoyer, se connecter et l'exécuter :

```
scp test pi@192.168.1.1:/home/pi/test
ssh pi@192.168.1.1
cd /home/pi/test
./test
```

On constate que le code s'exécute correctement. Lorsque l'on teste son exécution sur la machine virtuelle en revanche, cela ne fonctionne pas car il n'a pas été compilé pour ce type d'environnement.

## TP 3

### 1. Compilation du noyau

On récupère l'archive linux-rpi-3.10.y.zip sur le dépôt GitHub.

On copie et renomme le fichier « config-linux-qemu » par « .config » dans le dossier contenant l'archive décompressée.

On exécute la commande suivante afin de patcher le noyau :

```
Patch -p1 -d linux-rpi-3.10.y/ < Desktop/Noyau/linux-arm.patch
```

On exécute les commandes suivantes :

```
make ARCH=arm oldconfig  
make ARCH=arm menuconfig
```

On entre dans une interface graphique nous permettant de modifier les paramètres du fichier de configuration. On vérifie que l'outil de cross-compilation est bien « arm-linux-gnueabi- »  
Enfin on exécute les commandes suivantes pour finaliser le noyau :

```
cd  
export PATH=$(pwd)/raspbian-toolchain-gcc-4.7.2-linux64/bin:$PATH  
echo $PATH  
cd linux-rpi-3.10.y/  
make ARCH=arm
```

### 2. Test sur simulateur

On teste notre noyau sur le simulateur avec les commandes :

```
Apt-get install qemu-system-arm  
qemu-system-arm -kernel ./arch/arm/boot/zImage -cpu arm1176 -m 256 -M versatilepb  
-no-reboot
```

Afin de ne pas obtenir l'erreur du kernel panic, on cherche grâce à la commande suivante dans quel fichier elle est définie :

```
Grep -r 'Cannot open root device' ./*
```

Elle nous indique que cette ligne est présente dans le fichier `do_mounts.c`

On modifie donc ce fichier pour éviter le kernel panic en créant une boucle infinie avant la prise en charge de l'erreur.

On recompile le kernel avec la commande suivante afin que les modifications soient prises en compte :

```
Make ARCH=arm
```

Puis on lance le simulateur :

```
qemu-system-arm -kernel ./arch/arm/boot/zImage -cpu arm1176 -m 256 -M versatilepb  
-no-reboot
```

### 3. Test sur raspberry

On remplace le fichier `.config` par le fichier nommé « `config-linux-carte` » que l'on renomme « `.config` ». On exécute les commandes suivantes :

```
make ARCH=arm oldconfig  
make ARCH=arm menuconfig
```

On entre dans une interface graphique nous permettant de modifier les paramètres du fichier de configuration. On vérifie que l'outil de cross-compilation est bien « `arm-linux-gnueabi` ». On constate que ce n'est pas le cas et on change donc la valeur.

On recompile le kernel avec la commande suivante afin que les modifications soient prises en compte :

```
Make ARCH=arm
```

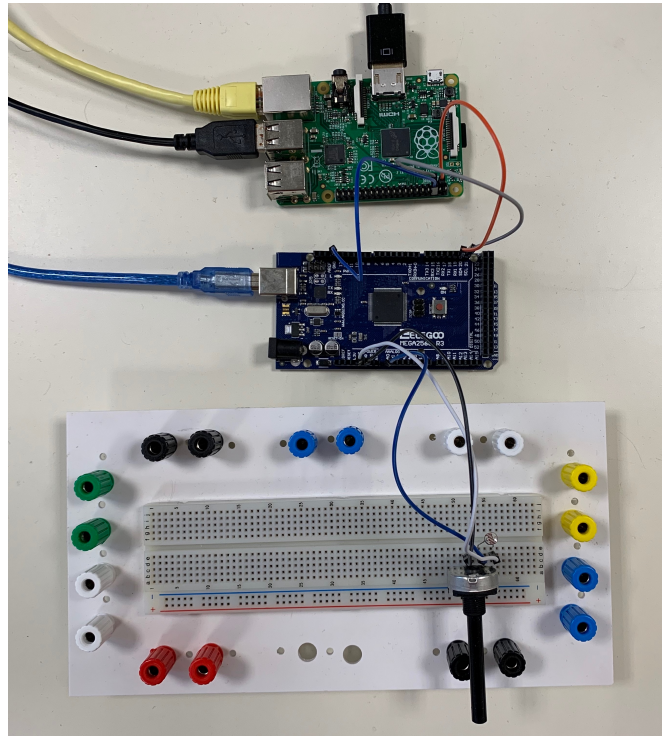
Lors de nos tests, le démarrage du noyau reste bloqué mais aucune erreur n'apparaît.

Lorsqu'on le branche avec l'adaptateur FTDI, des caractères apparaissent mais rien d'intelligible ne s'affiche.

# Projet

## 1. Création du montage

Pour ce projet, nous allons devoir mettre en place un système enfoui, c'est-à-dire autonome mais relié à une prise de courant, afin de récupérer des valeurs de luminosité. Pour cela, nous allons utiliser une photorésistance en guise de capteur que nous placerons dans un pont diviseur de tension. Nous récupérerons la tension grâce à une carte Arduino car pourvue d'entrée analogique contrairement à la raspberry pi. Les valeurs seront envoyées de l'Arduino à la raspberry pi par un bus I2C. Cette dernière traitera ces valeurs avant de les envoyer sur une base de données MongoDB dans le cloud (mLam.com). Enfin, une application Android récupérera ces valeurs, affichera la dernière et créera un graphique avec les autres. Comme indiqué plus haut, le système devra fonctionner dès son allumage sans configuration supplémentaire.



Sur cette photo, on peut voir (de haut en bas) la Raspberry pi, l'Arduino ainsi que le pont diviseur de tension. Nous avons utilisé un potentiomètre sur ce dernier afin de trouver plus facilement une valeur de résistance convenable. On commence par relier les deux cartes en I2C.

## 2. Configuration de l'Arduino

On écrit un programme afin que l'Arduino récupère la valeur produite par le capteur et l'envoie à la raspberry pi. On se rend compte que les données envoyées ne tiennent pas sur 1 octet et qu'il est donc nécessaire de trouver un moyen d'envoyer 2 octets de manière ordonné, pour ce faire on va utiliser la Raspberry Pi. Celle-ci va demander à l'Arduino de lire la valeur du capteur puis d'envoyer le premier bit puis le second.

*Voir Arduino/I2C\_slave\_raspberry.ino*

### 3. Configuration de la Raspberry Pi

On active ensuite l'I2C sur la Raspberry Pi en entrant dans la configuration de cette dernière :

```
sudo raspi-config
```

On se rend dans les options d'interfaces puis sur l'interface I2C qu'on active. On redémarre la carte afin que les changements soient pris en compte.

*Voici Raspberry/I2C\_master\_raspberry.py*

Pour pouvoir récupérer la valeur des deux octets, on fait en sorte que le premier octet contienne les valeurs les moins significatives et que le deuxième ne soit plus que le décalage vers la droite des bits de la valeur :

296 -> 0000 0001 0010 1000

1<sup>er</sup> octet : 0010 1000

2<sup>ème</sup> octet : 0000 0001 ~~0010 1000~~

On crée ensuite des cas auxquels la Raspberry Pi réagit différemment, si la valeur du deuxième octet est de 1 alors on incrémente la valeur du premier de 256 :


Valeur 2<sup>ème</sup> octet : 1

Valeur 1<sup>er</sup> octet : 40

Valeur mesurée = 40 + 256 = 296

Pour une valeur de 2, on ajoute 512 et une valeur de 3, 768.

On retrouve dans ce code une requête POST qui nous permet d'envoyer les données ainsi que la date et l'heure dans une base de données MongoDB.



[Home](#) : { db: 'matpau' }  
**Collection: mesure**

Documents

Indexes

Stats

Tools

Documents

Delete all documents in collection
Add document

Start new search

All Documents

Display mode:
list
table
(edit table view)

records / page
10
[ 1 - 10 of 19 ]
next > last >>

<pre>{   "_id": {     "\$oid": "5c488afd1f6e4f5f90cb12de"   },   "date": "2018/11/13",   "heure": "20:18:31",   "mesure": 747 }</pre>	
<pre>{   "_id": {     "\$oid": "5c488b0c1f6e4f5f90cb12f1"   },   "date": "2018/11/13",   "heure": "20:18:45",   "mesure": 718 }</pre>	
<pre>{   "_id": {     "\$oid": "5c488b1b5d0e651a02306c6d"   },   "date": "2018/11/13",   "heure": "20:19:0",   "mesure": 755 }</pre>	
<pre>{   "_id": {     "\$oid": "5c48937b5d0e651a02309951"   },   "date": "2018/11/13",   "heure": "20:54:45",   "mesure": 696 }</pre>	
<pre>{   "_id": {     "\$oid": "5c4893961f6e4f5f90cb5544"   },   "date": "2018/11/13",   "heure": "20:55:12",   "mesure": 717 }</pre>	
<pre>{   "_id": {     "\$oid": "5c4893a55d0e651a02309d9d"   },   "date": "2018/11/13",   "heure": "20:55:27",   "mesure": 698 }</pre>	
<pre>{   "_id": {     "\$oid": "5c4893b41f6e4f5f90cb55a5"   },   "date": "2018/11/13",   "heure": "20:55:41",   "mesure": 706 }</pre>	
<pre>{   "_id": {     "\$oid": "5c4893c31f6e4f5f90cb55be"   },   "date": "2018/11/13",   "heure": "20:55:56",   "mesure": 706 }</pre>	

Documents (aka Objects)

From the "Documents" tab you can browse and search for objects in this collection. All standard query constructs are supported except for `map/reduce` queries. To use `map/reduce`, use the MongoDB shell (note that temporary result collections will be viewable in mLab).

You can also add, edit, and delete individual documents from here. Bulk collection updates are not yet supported in this UI (although they are supported in the shell).

Afin que ce programme se lance à chaque démarrage de la carte sans intervention extérieure, on modifie le fichier `rc.local` en ajoutant la ligne suivante :

```
sudo python /home/pi/i2c.py &
```

## 4. Création de l'application Android

On crée ensuite une application Android afin de récupérer les données, les mettre en formes et les afficher :

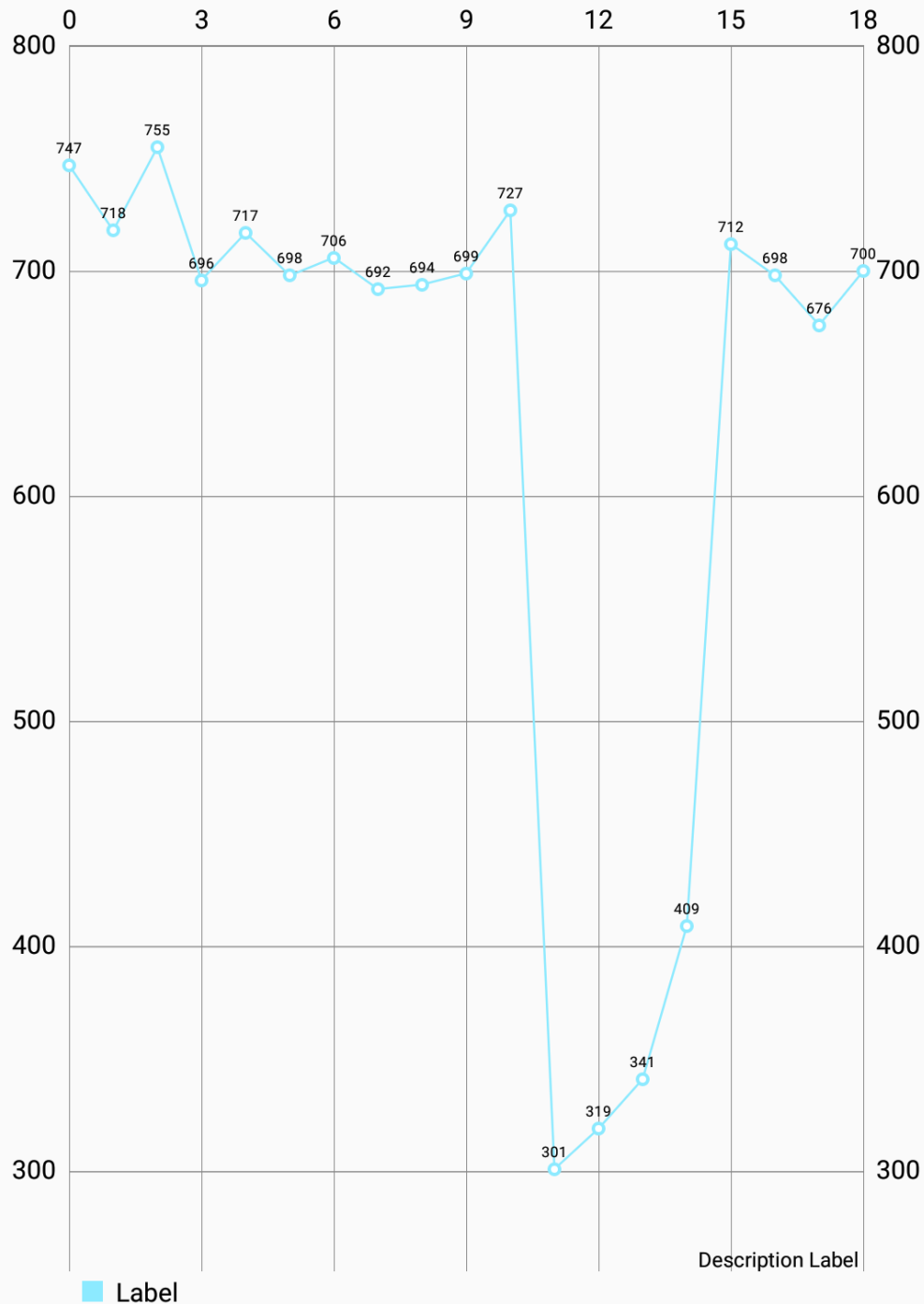




80B/s 4G 100% 17:12

# raspberry

2018/11/13 22:18:34 700



Voir [Android/app/src/main/](#)