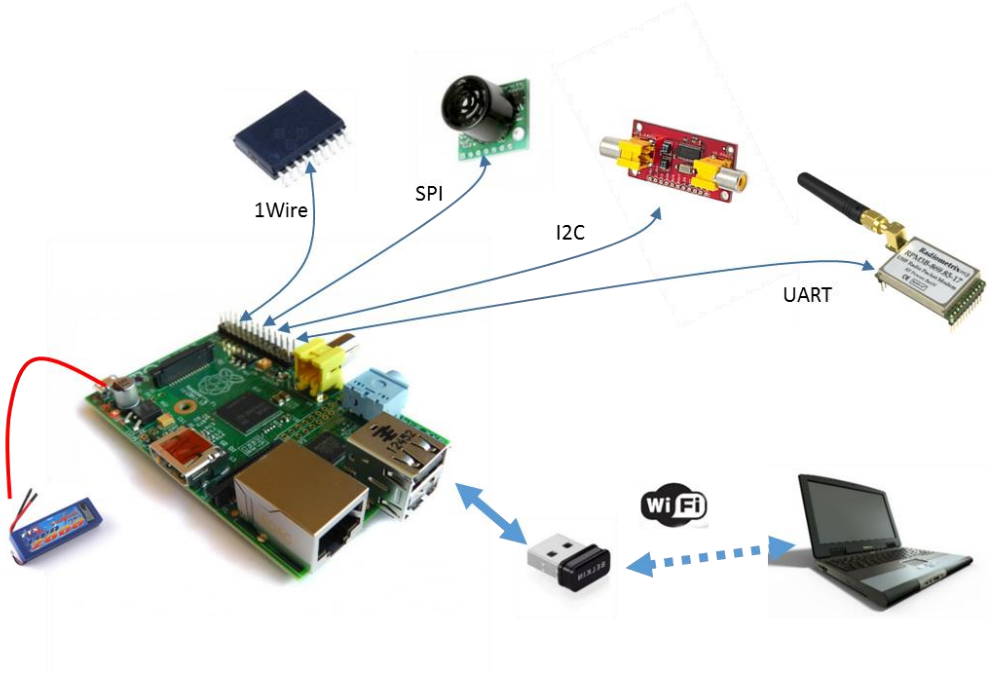
TP : Prise main d'un système embarqué Rapsberry Py



1 Introduction

Les ordinateurs embarqués sous le système d'exploitation Linux sont massivement présents dans les technologies modernes (transports, multimédia, téléphonie mobile, appareils photos …).

L'ordinateur Raspberry PI constitue un support d'apprentissage performant, très bon marché et disposant d'une forte communauté sur le net. Il possède des entrées/sorties puissantes permettant une connexion avec le monde physique par l'intermédiaire de capteurs et d'actionneurs.

L'objectif de ce TP est de réaliser une rapide prise en main d'un système embarqué au travers d'un ordinateur Raspberry PI et d'effectuer un tour d'horizon des pratiques de mise en œuvre et de développement.

Les connaissances acquises devront permettre la mise en œuvre d'un objet connecté simplifié.

2 Evaluation

L'évaluation portera sur :

- Un compte rendu détaillé explicitant les différentes opérations menées durant le TP.
  Pensez à prendre des copies d'écran pour illustrer ces opérations.
- la réalisation d'un objet connecté qui reprendra les méthodes de travail et l'utilisation des bus de communication abordés dans le TP.

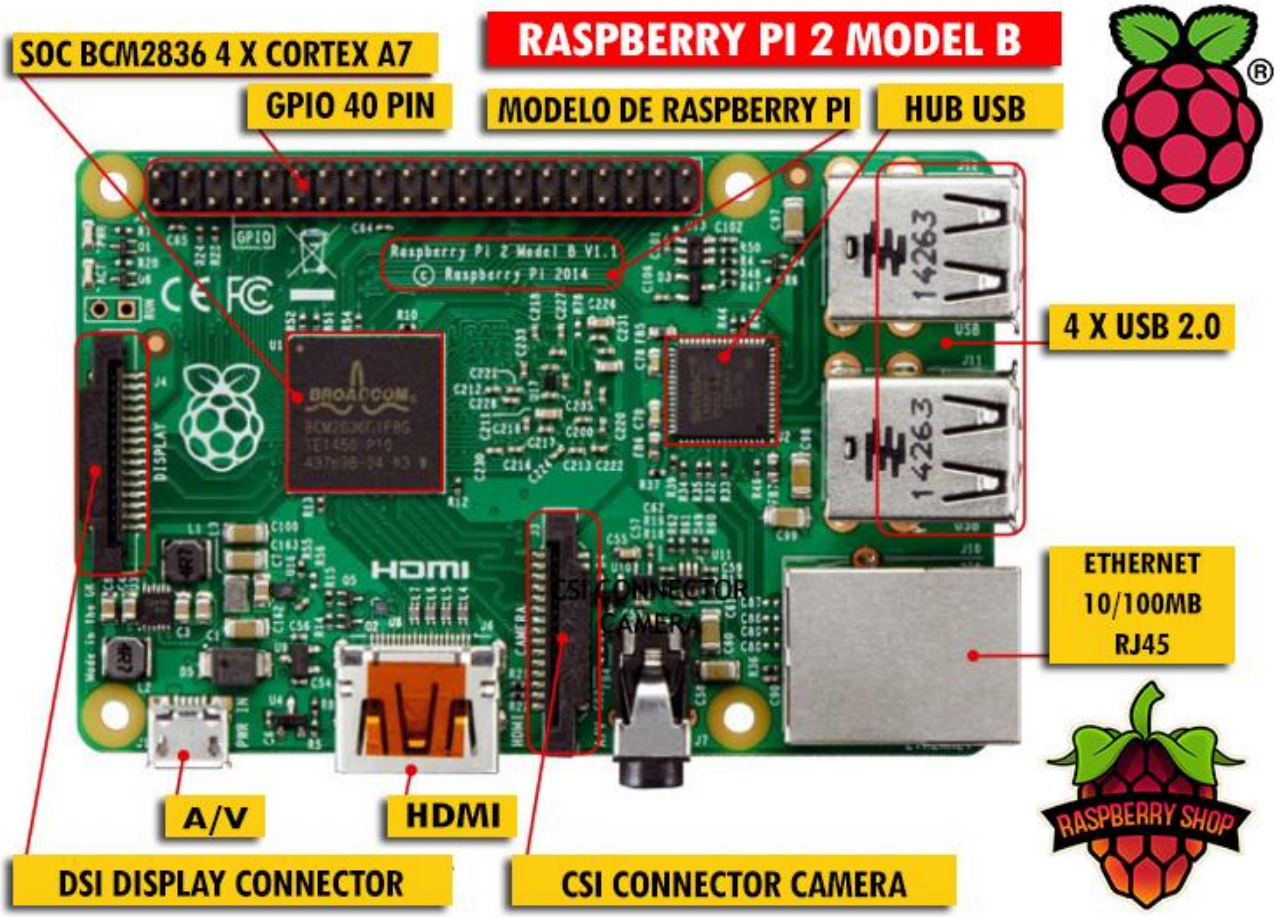
3 Description de l'ordinateur embarqué

3.1 Présentation

Raspberry Pi est un petit ordinateur sous le système d'exploitation Linux sur carte SD destiné à des applications d'informatique embarquée. Le cœur de l'ordinateur est un FPGA (Broadcom 2836) intégrant un processeur quad-core ARM Cortex A7 cadencé à 900MHz, 1Go de RAM et de nombreux périphériques.

Raspberry Pi peut être directement connecté à une IHM classique, souris/clavier/ecran HDMI ou vidéo composite, cependant comme tout ordinateur Linux, Raspberry Pi peut intégrer ses propres outils de développement et une IHM reposant sur SSH contrôlable depuis un autre ordinateur par Ethernet ou WIFI.

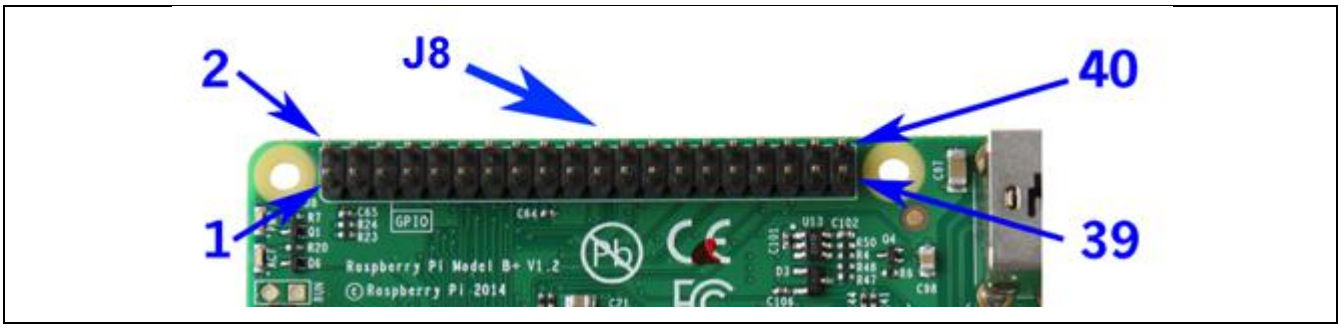
Le connecteur d'extension supporte les entrées/sorties parallèles ainsi que la plupart des bus de communication. C'est un support particulièrement économique et puissant qui peut être facilement mis en œuvre dans de petits systèmes nécessitant un accès au monde physique par des capteurs/actionneurs disposants d'interfaces numériques.



3.2 Les connecteurs d'extension

Le connecteur d'extension de la carte Raspberry PI est utilisé pour raccorder des périphériques de communication (UART, I2C, SPI, ...) ou TOR (Tout Ou Rien).

Les broches peuvent avoir des fonctions différentes suivant qu'elles sont activées en tant que GPIO (Global Purpose Input Output), périphérique de communication ou sorties PWM (Pulse Width Modulation).



Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

http://www.element14.com/community/docs/DOC-73950/l/raspberry-pi-2-model-b-gpio-40-pin-block-pinout

3.3 Raspbian

Raspbian est un système d'exploitation libre basé sur la distribution GNU/Linux Debian, et optimisé pour le plus petit ordinateur du monde, la Raspberry Pi.

Raspbian ne fournit pas simplement un système d'exploitation basique, il est aussi livré avec plus de 35 000 paquets, c'est-à-dire des logiciels pré-compilés livrés dans un format optimisé, pour une installation facile sur votre Raspberry Pi via les gestionnaires de paquets.

La Raspberry Pi est une framboise merveilleuse, mais elle reste néanmoins dotée d'une puissance inférieure à celle d'un ordinateur moderne. Par conséquent, il est préférable d'installer un système optimisé pour la Raspberry.

Raspbian a été créé dans cette optique, et il est donc tout particulièrement adapté à la Raspberry.

Par ailleurs, en tant que distribution dérivée de Debian, il répond à la majeure partie de la très vaste documentation de Debian.

https://www.raspbian.org/

Vous disposez de l'image 2018-11-13-raspbian-stretch.img dans le dossier Documents/TP-Raspbian.

déployez cette image sur la carte microSD à votre disposition en utilisant l'utilitaire **Etcher** (il faudra probablement l'installer) ou en utilisant la commande `dd` :

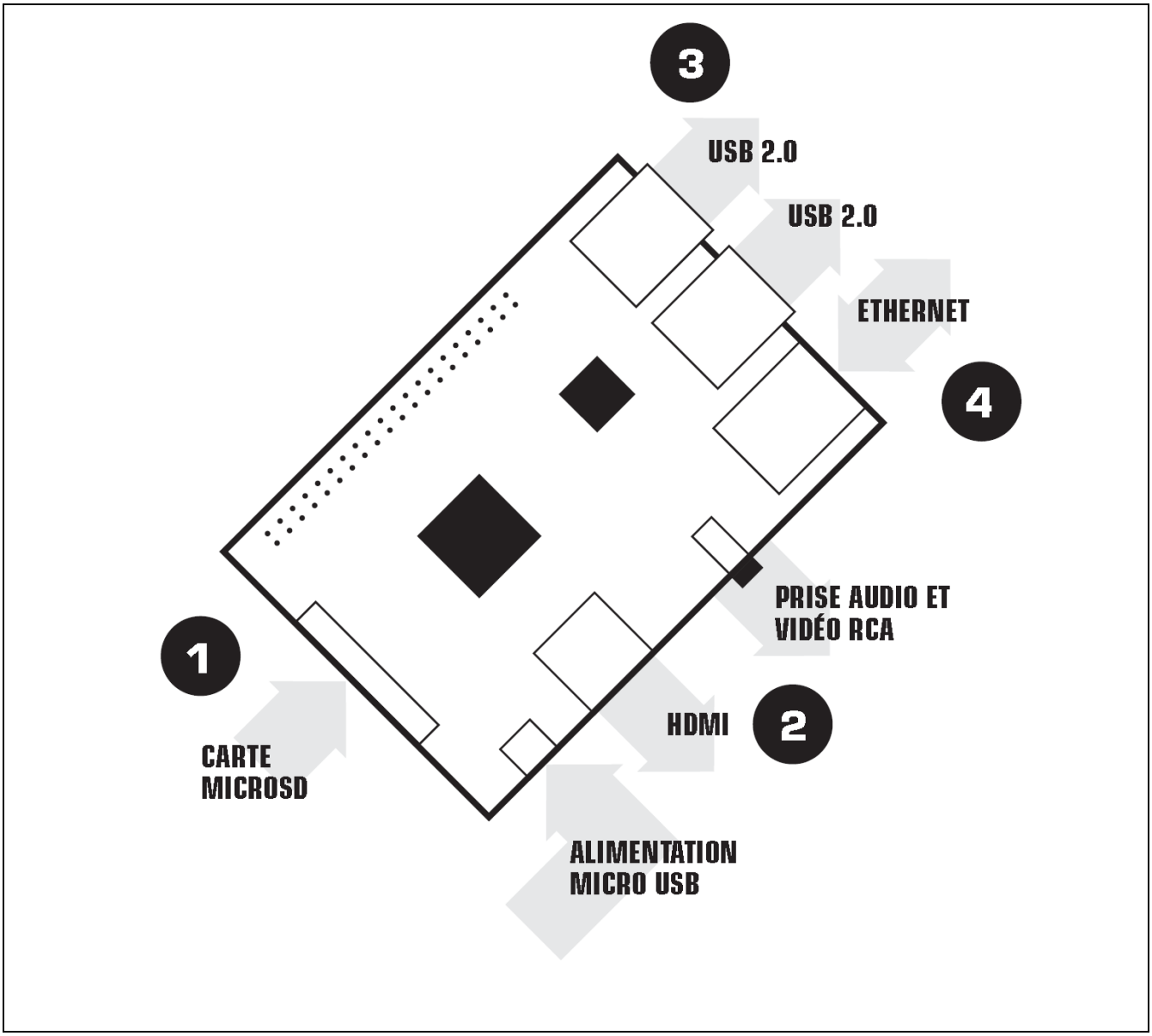
```
$ sudo dd if=2018-11-13-raspbian-stretch.img | pv | sudo dd of=/dev/sdx bs=8192
```

Attention : Remplacez x dans sdx par la lettre de média appropriée !

Le processus de déploiement est long (au moins 10 min).

4 Prise en main directe

4.1 Connexion des périphériques

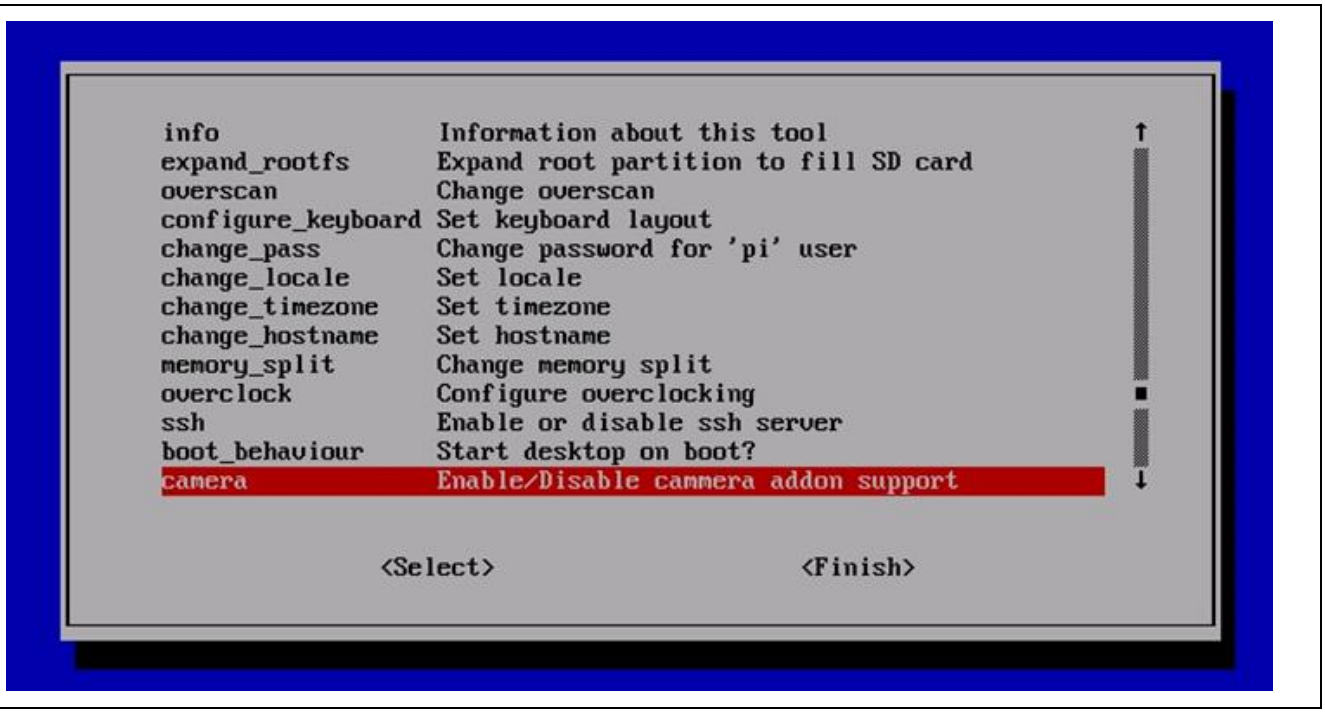


Avant de brancher quoi que ce soit à votre Raspberry Pi, assurez-vous de disposer de l'ensemble des éléments listés. Puis, suivez ces instructions :

- Commencez par insérer votre carte microSD dans le logement prévu sur le Raspberry Pi.
- Ensuite, connectez votre souris et votre clavier USB aux ports USB du Raspberry Pi.
- Assurez-vous que votre moniteur est sous tension et que vous avez sélectionné l'entrée appropriée (p. ex. HDMI 1, DVI, etc.).
- Ensuite, reliez votre Raspberry Pi à votre moniteur à l'aide du câble HDMI.
- Connectez votre Raspberry Pi au réseau à l'aide d'un câble Ethernet.
- Lorsque vous avez branché tous les câbles et inséré la carte microSD requise, branchez l'alimentation électrique micro USB. Cette action met sous tension et démarre votre Raspberry Pi.

4.2 Démarrage

- Au premier démarrage, le système de fichier est redimensionner pour occuper tout l'espace disponible sur la carte microSD.
- Un assistant vous invite à configurer le pays, la langue et la zone de temps.
- **Ne modifiez pas le mot de passe par défaut de l'utilisateur « pi » : « raspberry »**
- Activer le serveur ssh à l'aide de l'utilitaire de configuration : **`sudo raspi-config`**



La mise à l'heure nécessite une connexion à un serveur NTP, elle ne sera effective que lorsque la connexion à internet sera bien établie. Pour cela, vous devrez vous identifier via le portail captif du l'université d'Avignon.

4.3 Connexion au réseau

Pour se connecter à un réseau, il faut renseigner le fichier **interfaces** : **/etc/network/interfaces**

- Configurez la connexion filaire pour recevoir automatiquement les paramètres TCP/IP d'un DHCP.
- Identifiez votre adresse IP.
- Vérifiez dans le navigateur l'accès à internet.

4.4 Mise à jour

- Mise à jour de la liste des paquets
  apt-get update
- Mise à jour des paquets installés **(NE PAS FAIRE ICI !)**
  apt-get upgrade
- Mise à jour de la distribution Raspbian **(NE PAS FAIRE ICI !)**
  apt-get dist-upgrade

4.5 Compilation sur cible

C'est le mode de compilation le plus simple : on procède comme sur un PC fonctionnant sous une distribution classique le linux.

- Saisissez le programme suivant en enregistrez dans le dossier personnel de l'utilisateur **pi** sous le nom **hello1.cpp** :

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
        string prenom;
        cout << "Test d'ecriture et de compilation d'un programme C++ sur Raspbery PI " << endl ;
        cout << "- Ecriture sur cible  " << endl ;
        cout << "- Enregistrement local  " << endl ;
        cout << "- Compilation sur cible  " << endl ;
        cout << "Quel est ton prenom ? " ;
        cin >> prenom;
        cout << "Bonjour " << prenom << endl;
}
```

- Compilez et exécutez le programme. Sauvegardez le rapport de compilation.
  g++ -ftime-report hello1.cpp –o hello1

5 Prise en main à distance

5.1 ssh / scp

En général, les systèmes embarqués n'ont pas de moniteur, de clavier et de souris, ils sont conçus pour fonctionner à l'intérieur d'un système matériel et donc inaccessible. Leur gestion se fait alors à distance, par l'intermédiaire de liaisons de communications séries (souvent RS232) ou par réseau filaire ou wifi. On accède alors au système en mode console grâce au protocole SSH (Secure Shell) et on gère les fichiers grâce à l'utilisation de protocoles d'échange comme FTP, SFTP ou encore SCP. On peut également gérer l'interface graphique en mode distant (« terminal distant ») mais l'utilisation d'une interface graphique dans la plupart des systèmes embarqués est une hérésie. Elle consomme des ressources inutilement.

5.2 Hôte linux

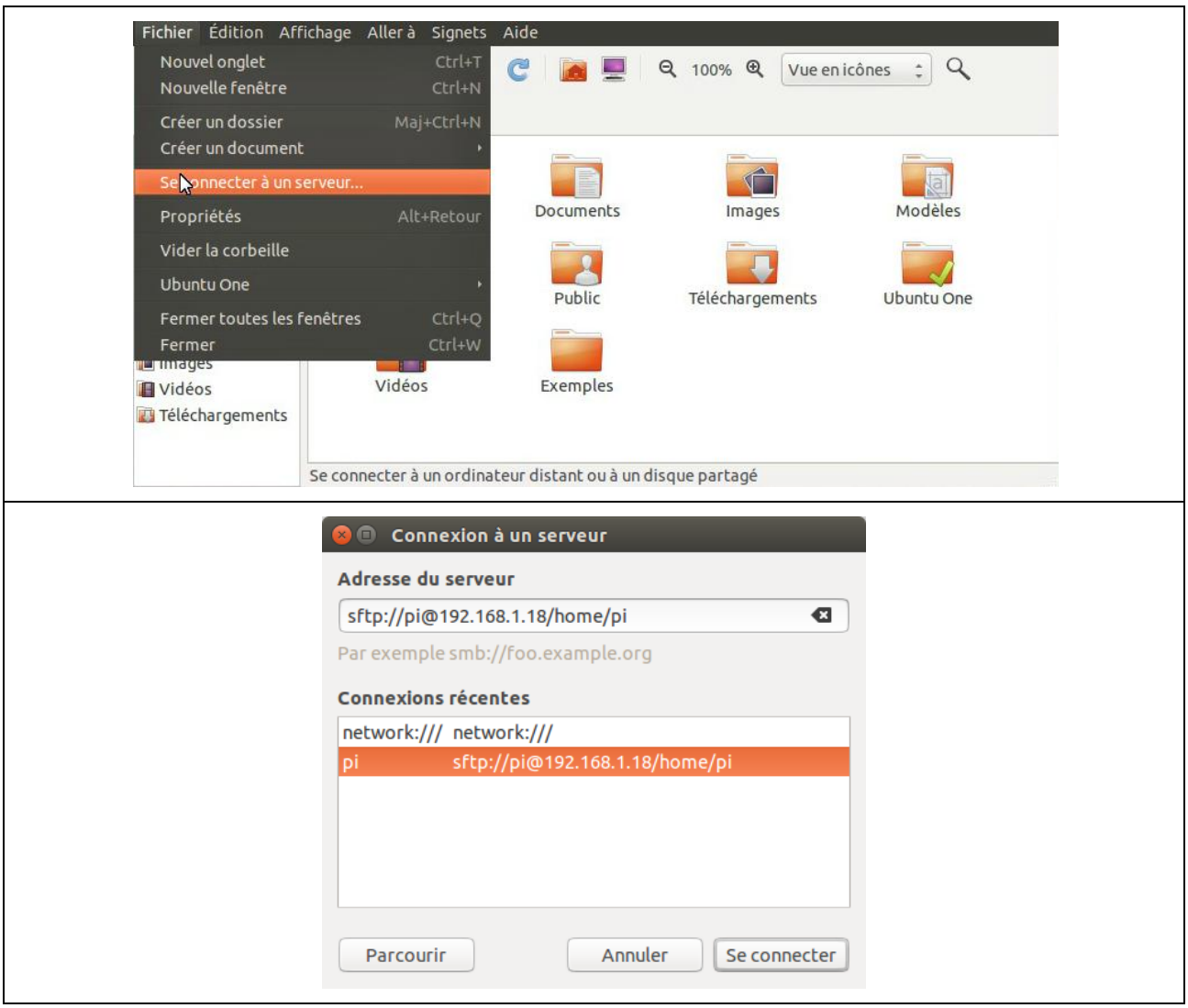
Sur un hôte linux standard, les protocoles SSH et SCP sont généralement déjà installé.

- Depuis un hôte linux connecté au réseau, ouvrez une session SSH avec la Raspberry PI dans un terminal. Explicitez les différentes étapes.
  ssh <IP_Raspberry_PI>

- La commande **scp** permet de copier un fichier ou un répertoire (**-r**) d'un client vers un serveur ou d'un serveur vers un client. Depuis votre hôte linux, copiez le fichier **hello1.cpp** présent sur la Raspberry PI dans votre dossier personnel.
  scp <Fichier_local> <login@IP_client_distant:Chemin>
  scp <login @IP_serveur_distant:Chemin/Fichier> <Chemin/Fichier_local>

  Explicitez les différentes étapes et donnez la syntaxe précise utilisée.

- Le gestionnaire de fichiers de l'hôte local est Nautilus. Ce gestionnaire de fichiers permet de se connecter à un serveur distant et de monter le dossier indiqué dans le système de fichiers local, se qui permet d'y naviguer graphiquement. Depuis l'hôte linux, ouvrez Nautilus :



Remarque : Utilisez l'adresse IP de votre carte Raspberry PI.

5.3 Cross-compilation

Le processus de compilation mobilise les ressources du système (temps processeur et mémoire). Il nécessite également un espace de stockage suffisant pour l'enregistrement des données temporaires de compilation.

Sur un système embarqué, ces caractéristiques matérielles sont naturellement limitées et il est souvent préférable de procéder à la compilation des sources sur un système plus performant puis de transférer les exécutables obtenus sur la cible.

Ce type de compilation s'appelle cross-compilation ou compilation croisé en français. Il s'agit de compiler sur une machine (PC) pour une autre (Raspberry PI). Le cross-compilateur doit être construit sur mesure, pour une cible donnée et sur un système hôte donné. Il faut donc le compiler en utilisant le compilateur de l'hôte, des outils pour la cible (**binutils**) et des bibliothèques C adaptées (**libC**). Cette préparation peut être très complexe. Il faut faire les bons choix et résoudre les problèmes de dépendances. Heureusement, il existe des solutions « pré-faites » comme **crosstool-NG**. Il permet de fabriquer sur mesure les outils nécessaires pour faire de la compilation croisée, en l'occurrence, les outils nécessaires pour compiler sur un PC x86-64 pour une plate-forme ARM.

Les distributeur d'OS mettent également à disposition des développeurs des solutions de cross-compilation. C'est le cas de Raspbian pour le Raspberri Py :
https://www.raspberrypi.org/documentation/linux/kernel/building.md

5.3.1 Installer la chaine de compilation croisée

Suivez les instructions d'installation de la chaine de compilation croisée pour Raspberri Py :

- https://www.raspberrypi.org/documentation/linux/kernel/building.md
- INSTALL TOOLCHAIN

5.3.2 Compilation et test

- Ouvrez l'éditeur de texte de votre hôte linux et saisissez et enregistrez localement le programme suivant sous le nom **hello2.cpp** :

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string prenom;
   cout << "Test d'ecriture et de cross compilation d'un programme C++ pour Raspbery PI" << endl ;
   cout << "- Ecriture sur hote distant Ubuntu  " << endl ;
   cout << "- Transfert par ssh / serveur monte dans FS Ubuntu  " << endl ;
   cout << "- Cross compilation sur hote  " << endl ;
   cout << "Quel est ton prenom ? " ;
   cin >> prenom;
   cout << "Bonjour " << prenom << endl;
}
```

- Depuis l'hôte linux, compilez, transférez l'exécutable généré et exécutez le sur la cible. Sauvegardez le rapport de compilation.

  arm-linux-gnueabihf-g++ -ftime-report hello2.cpp -o hello2

- Explicitez les différentes étapes.

6 Utilisation des GPIO

Le Raspberry Pi offre quelques possibilités d'entrées-sorties directes en utilisant les broches GPIO présentes sur son connecteur P1. Elles ne sont pas très nombreuses (une dizaine) mais cela peut suffire pour des petits projets interactifs nécessitant d'interroger des capteurs tout-ou-rien ou de valider des actionneurs.

Nous pouvons utiliser ces GPIO de différentes façons, depuis l'espace utilisateur ou depuis le noyau. Le but ici est de balayer les principaux aspects.

6.1 L'interface sysfs

Sysfs est un système de fichiers virtuel introduit par le noyau Linux 2.6. Sysfs permet d'exporter depuis l'espace noyau vers l'espace utilisateur des informations sur les périphériques du système et leurs pilotes, et est également utilisé pour configurer certaines fonctionnalités du noyau.

Pour chaque objet ajouté dans l'arbre des modèles de pilote (pilotes, périphériques, classes de périphériques), un répertoire est créé dans sysfs.

- La relation parent/enfant est représentée sous la forme de sous-répertoires dans **/sys/devices/** (représentant la couche physique).
- Le sous-répertoire **/sys/bus/** est peuplé de liens symboliques, représentant la manière dont chaque périphérique appartient aux différents bus.
- **/sys/class/** montre les périphériques regroupés en classes, comme les périphériques réseau par exemple.

Pour les périphériques et leurs pilotes, des attributs peuvent être créés. Ce sont de simples fichiers, la seule contrainte est qu'ils ne peuvent contenir chacun qu'une seule valeur et/ou n'autoriser le renseignement que d'une valeur. Ces fichiers sont placés dans le sous-répertoire du pilote correspondant au périphérique.

6.2 Lire/Ecrire sur une broche TOR

6.2.1 Accès depuis l'espace utilisateur

L'accès simple, depuis le shell peut se faire très aisément grâce au système de fichiers **/sys**.

```
/ # cd /sys/class/gpio/
/sys/class/gpio # ls
export     gpiochip0  unexport
```

Accédons au GPIO 24 (broche 18) :

```
/sys/class/gpio # echo 24 > export
/sys/class/gpio # ls
export     gpio24     gpiochip0  unexport
/sys/class/gpio # cd gpio24/
/sys/class/gpio/gpio24 # ls
active_low   direction   edge   subsystem   uevent   value
```

6.2.2 Ecriture

La broche doit être configurée en sortie pour écrire :

```
/sys/class/gpio/gpio24 # echo out > direction
/sys/class/gpio/gpio24 # echo 1 > value
/sys/class/gpio/gpio24 # echo 0 > value
```

6.2.3 Lecture

La broche doit être configurée en entrée pour lire :

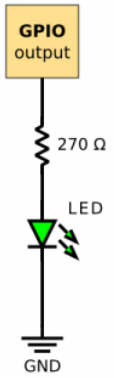
```
/sys/class/gpio/gpio24 # echo in > direction
/sys/class/gpio/gpio24 # cat value
0
```

6.2.4 Programmation des GPIO

L'utilisation des ports GPIO est liée à l'accès à plusieurs fichiers. Les opérations d'export, de configuration de la direction, de lecture ou d'écriture peuvent être réalisées dans n'importe quel langage de programmation.

6.3 Application

- Complétez l'application C++ permettant de faire clignoter une LED connectée sur la broche 18 avec une fréquence réglable lorsque l'on appui sur un bouton poussoir que vous connecterez sur un port GPIO de votre choix (proposez un schéma de connexion).
- Cross compilez et déployez l'exécutable sur le Raspberry Pi.
- Exécutez et faites valider le fonctionnement.



```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <string>

// Remplacez les ... par le code approprié

#define LED24 "/sys/class/gpio/ ..."
#define BUTTON "/sys/class/gpio/ ... "

using namespace std;

int main(...)
{
        ofstream led24;
        ifstream button;

        while(1)
        {
                button.open(...);
                string buttonState;
                getline(button,buttonState);
                button23.close();
                if(buttonState=="1")
                {
                        led24.open(LED24);
                        //Allumer
                        led24 << "...";
                        led24.close();
                        sleep(...);
                        led24.open(LED24);
                        //Eteindre
                        led24 << "...";
                        led24.close();
                        sleep(...);
                }
        }
}
```

7 Utilisation du bus I2C

7.1 Le bus I2C sur la Raspberry pi

Le Raspberry Pi dispose de deux interfaces i2c. Le bus numéro 0 qui était accessible à travers le connecteur P5 sur le modèle B a disparu depuis le modèle B+. De plus, ce connecteur étant présent sous forme de simples trous cuivrés, il était donc nécessaire de venir y souder des broches pour pouvoir l'utiliser. Les signaux du bus 0 sont toujours présents – mais pas très accessibles – sur les connecteurs J3 (camera) et J4 (display).

Nous allons nous intéresser au second bus, accessible via le port d'extension P1, sur deux broches (que l'on peut également employer pour des entrées/sorties GPIO) identiques quel que soit le modèle de Raspberry. Il s'agit de la broche 3 (signal SDA) et de la broche 5 (signal SCL).

- Rappelez à quoi servent ces deux signaux.

7.2 Activation du bus I2C sur le Raspberry

Le protocole i2c est supporté par le noyau Linux depuis sa version 2.4. De nombreux périphériques sont reconnus par le kernel, notamment dans le sous-système Hwmon (Hardware Monitor).

L'accès depuis l'espace utilisateur est facilité par le module i2c-dev qui rend les bus i2c visibles dans le répertoire /dev sous forme de fichiers spéciaux représentant des périphériques en mode caractère.

- Activez la prise en charge du bus I2C :

```
$ sudo raspi-config
```

-> Advanced options -> I2C -> Enabled (tout accepter).

Vérifiez la prise charge du bus I2C, identifiez la vitesse de transmission sur le bus :

```
$ dmesg | grep i2c
```

Identifiez le(s) bus i2c-c disponible(nt) :

```
$ ls /dev/i2c*
```

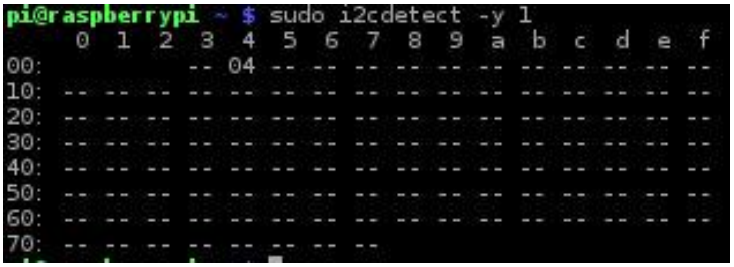
Installez éventuellement les outils de gestion du bus i2c :

```
$ sudo apt-get install i2c-tools
```

Détectez des esclaves sur le bus :

```
$ i2cdetect  -y  1
```

Remarque : *-y pour répondre automatiquement yes et 1 pour le numéro de périphérique i2c*

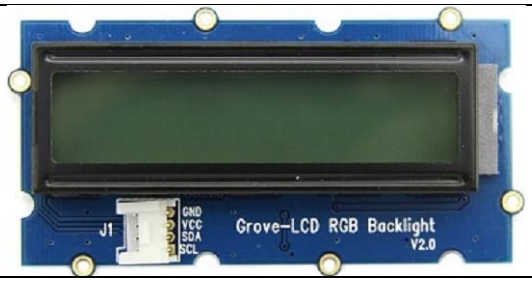


- Les commandes `i2cset` et `i2cget` permettent respectivement d'écrire et de lire sur le bus i2c.

7.3 Gestion d'un afficheur LCD RGB i2c

L'afficheur à gérer est un JHD1313M1 de chez Grove. Il est en fait composé d'un afficheur LCD basé sur le HD44780 d'Hitachi et d'un contrôleur du rétro-éclairage (RGB). Il possède donc 2 adresses i2c :

- LCD : 0x3e
- RGB : 0x62



- Connectez l'afficheur au connecteur de la carte Raspberry pi.
  Attention, l'afficheur nécessite une tension de 5V pour fonctionner correctement.
- Vérifiez la présence de l'afficheur sur le bus i2c.
- La commande i2cset s'utilise de la manière suivante :

  i2cset [**-f**] [**-y**] [**-m mask**] [**-r**] *i2cbus chip-address data-address* [*value*] ... [*mode*]

  Procédez à la séquence d'initialisation de l'afficheur :

```
i2cset -y 1 0x3E 0x80 0x3C
i2cset -y 1 0x3E 0x80 0x0C
i2cset -y 1 0x3E 0x80 0x01
i2cset -y 1 0x3E 0x80 0x06
i2cset -y 1 0x62 0x00 0x00
i2cset -y 1 0x62 0x08 0xFF
i2cset -y 1 0x62 0x01 0x20
i2cset -y 1 0x62 0x04 0x00
i2cset -y 1 0x62 0x03 0x00
i2cset -y 1 0x62 0x02 0xFF
i2cset -y 1 0x3E 0x40 0x31
i2cset -y 1 0x3E 0x40 0x32
i2cset -y 1 0x3E 0x40 0x33
```

- Identifiez les instructions passées à l'afficheur LCD (adresse i2c 0x3E) et au contrôleur de rétro-éclairage (adresse i2c 0x62). Identifiez les registres de gestions des couleurs :
  - datasheet afficheur LCD JHD*
  - datasheet contrôleur RGB PCA9633
- Configurez le rétro-éclairage pour qu'il clignote à une fréquence de 1Hz avec un rapport cyclique de 50%.
- Vous disposez dans le dossier **TP-Raspbian/fichiers/lcd** d'un programme de mise en œuvre basic de l'afficheur. Procédez à la cross-compilation du programme, transférez l'exécutable sur la cible et exécutez le.
- Modifiez le programme à votre guise et testez le résultats.
- Identifiez la fonction de bas niveau qui permet d'écrire sur le bus i2c.
  Quels appels système utilise-elle ?

8 Utilisation du bus one wire

8.1 Le bus one wire sur la Raspberry pi

Le bus 1-Wire (ou OneWire) est un bus conçu par Dallas Semiconductor.

Le niveau de tension utilisé sur ce bus est +5V (niveau TTL). Il supporte une topologie série, parallèle ou en étoile et fonctionne suivant le principe maître / esclave. Dans notre cas, c'est la Raspberry qui joue le rôle du maître.

L'avantage de ce bus est qu'il peut être utilisé en mode "parasite" (alimentation à partir du fil de données). Cela permet d'utiliser seulement 2 fils (et non un seul comme le nom le laisse supposer), un fil de données et un fil de masse.

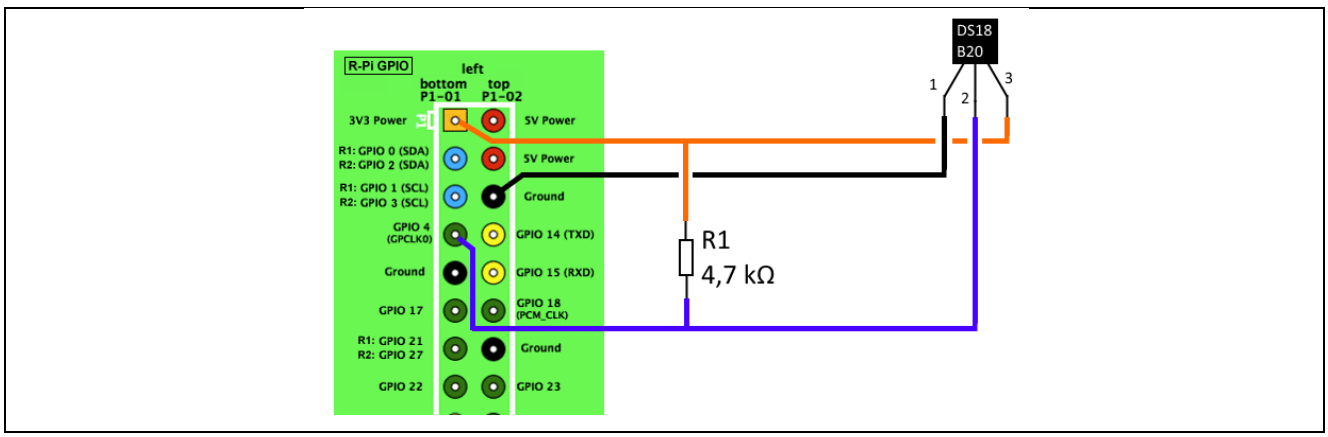
Généralement utilisé pour des mesures de températures, il existe une gamme complète de composants compatibles.

Chaque circuit possède une adresse physique unique, gravée dans la puce à la fabrication.

Linux dispose de drivers pour accéder à ce bus. Il faut installer les modules **w1-gpio** et **w1-therm**. Raspbian permet d'installer ces modules via l'interface de configuration `raspi-config`.

L'installation de ces modules monte dans le système de fichiers une nouvelle arborescence **/sys/bus/w1/devices** dans laquelle on trouve un dossier portant comme nom l'adresse d'un périphérique connecté au bus 1-wire. C'est le fichier **w1_slave** qui contient la donnée.

Au repos, le bus est à l'état haut. Il est nécessaire de fixer cet état au moyen d'une résistance de rappel.



8.2 Mesure de la température

Le capteur que nous allons utiliser ici est un capteur de température numérique à résolution programmable 1-Wire Maxim DS18B20. Encapsulé dans son boîtier TO-92, il présente un aspect similaire à beaucoup de transistors et comporte trois pattes (figure ci-dessus) : GND (masse), DQ (données) et VDD (alimentation à 3,3 V). Il peut aussi prendre l'aspect d'une sonde industrielle similaire aux traditionnelles PT100.

Interrogation du capteur :

```
/sys/bus/w1/devices $ ls
28-0000054c2ec2
/sys/bus/w1/devices $ cd 28-0000054c2ec2
/sys/bus/w1/devices/28-0000054c2ec2 $ ls
driver id name power subsystem uevent w1_slave
/sys/bus/w1/devices/28-0000054c2ec2 $ cat w1_slave
7c 01 4b 46 7f ff 04 10 09 : crc=09 YES
7c 01 4b 46 7f ff 04 10 09 t=23750
```

- Complétez l'application C++ `w1.cpp` permettant de lire la température toute les secondes.
- Cross-compilez, déployez l'exécutable sur la cible et exécutez le. Faites valider le fonctionnement.

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <string>
#include <cstdlib>
#define SENSOR "/sys/bus/w1/devices/..."

using namespace std;

int main()
{
        string line;
        string temp="";
        float temperature;

        while(1)
        {
                ifstream sensor(SENSOR);
                if (sensor.is_open())
                {
                        while ( getline (sensor,line) )
                        {
                                temp+=line;
                        }
                        sensor.close();
                        int pos = temp.find("t=");
                        if(pos>0)
                        {
                                temp=temp.substr(...);
                                temperature = stof(temp);
                                cout << temperature/... << "°C" << endl;
                        }
                        else cout << "Impossible de lire la temperature !" << endl;
                }
                else cout << "Lecture sensor impossible" << endl;
        }
}
```

9 Objet connecté

Le but de cette partie est de réaliser un objet connecté possédant une interface locale et une interface web de supervision distante.

Cet objet sera un thermomètre connecté. Les sources à compiler et à déployer sont disponibles sur les dépôts ci-dessous :

https://github.com/msilanus/RaspberryLcdTemp

Analysez le programme principale main.cpp et explicitez le fonctionnement de l'application.

9.1 Interface web

Vous pourrez utiliser une des solutions suivantes :

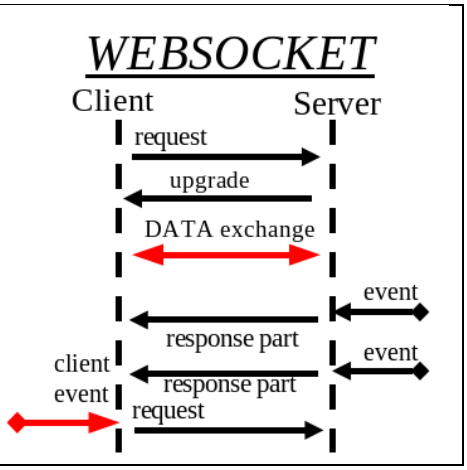
9.1.1 Serveur web

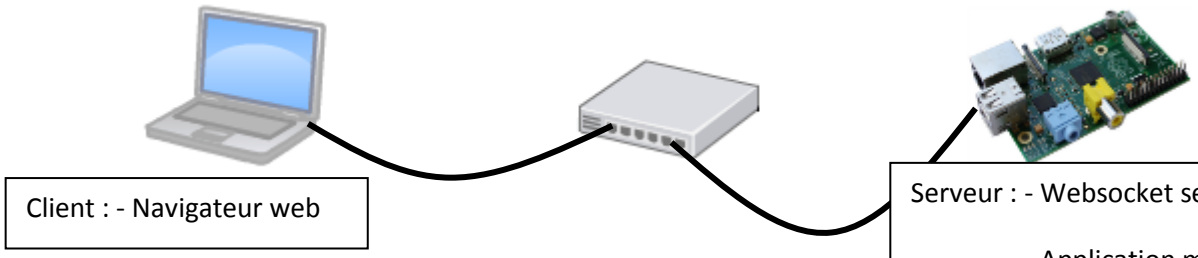
L'interface web pourrait être mise à disposition par un serveur http embarqué. De nombreuses solutions sont disponibles mais elles doivent prendre en compte de nombreuses problématiques liées à leurs caractéristiques. En effet, la taille réduite, la consommation énergétique, l'architecture flexible ainsi que le coût de ces serveurs sont autant de problématiques qu'il est nécessaire de soulever pour amener internet vers les systèmes embarqués.

La solution la plus utilisée, lorsque les performances du système le permettent, est **Apache** (environ 60% de part de marché). Pour une application aussi simple, **nginx** ou **lighttpd** sont parfaitement adaptés.

9.1.2 WebSockets

Une autre solution consiste en l'utilisation d'un serveur Websockets :

WebSocket est une spécification d'un protocole permettant une communication bidirectionnelle et full duplex sur une seule socket TCP entre un client et un serveur. Initialement développé pour HTML 5, WebSocket a été normalisé par l'IETF et le W3C. Tous les navigateurs récents implémentent et supportent les WebSockets. Ce protocole permet notamment d'implémenter facilement et de manière standard l'envoi de données en mode Push à l'initiative du serveur. http://tools.ietf.org/html/rfc6455	



- **Websocket serveur** : http://websocketd.com/
- **Application** : A développer dans le langage de votre choix.
- **Page web** : HTML et ou php / CSS / javascript

- L'application de mesure et d'affichage locale s'occupe de lire toute les secondes la température ou la distance à un obstacle et de les fournir aux deux IHM. Dans le cas de l'utilisation d'un serveur websocket, ce dernier se substitue aux flux standard d'entrée et de sortie.
- La page web doit contenir la connexion au websocket et les fonctions d'affichage :

```
<!DOCTYPE html>
<script>
  var msg;
  function log(msg)
  {
        // Ajouter une ligne de log : mesure horodatée (jj/mm/aaaa - hh:mm:ss | mesure unité)
  } ;
  function mesure(msg)
  {
        // Afficher la mesure
  } ;

  // Ouverture du websocket
  var ws = new WebSocket('ws://192.168.1.31:8080/');
  ws.onopen = function()
  {
    log('CONNECT');
  };
  ws.onclose = function()
  {
    log('DISCONNECT');
  };
  ws.onmessage = function(event)
  {
    msg = event.data;
    mesure(msg)
    log(msg);
  };
</script>
<HTML>
<HEAD>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <TITLE>Mon objet connect&eacute;</TITLE>
</HEAD>
<BODY>
<center><h1><div id='mesure'></div></h1></center>

<h2>   Historique :</h2>
<div id='log'></div>
</BODY>
</HTML>
```

Ce script est à placer dans le dossier indiqué par l'option - -staticdir lors de l'exécution du serveur. Le client devra spécifier explicitement le port, ici : http://192.168.1.31:8080

- Lancement du serveur websocket

```
$ ./websocketd --port=8080 --staticdir=. ./monAppli
```

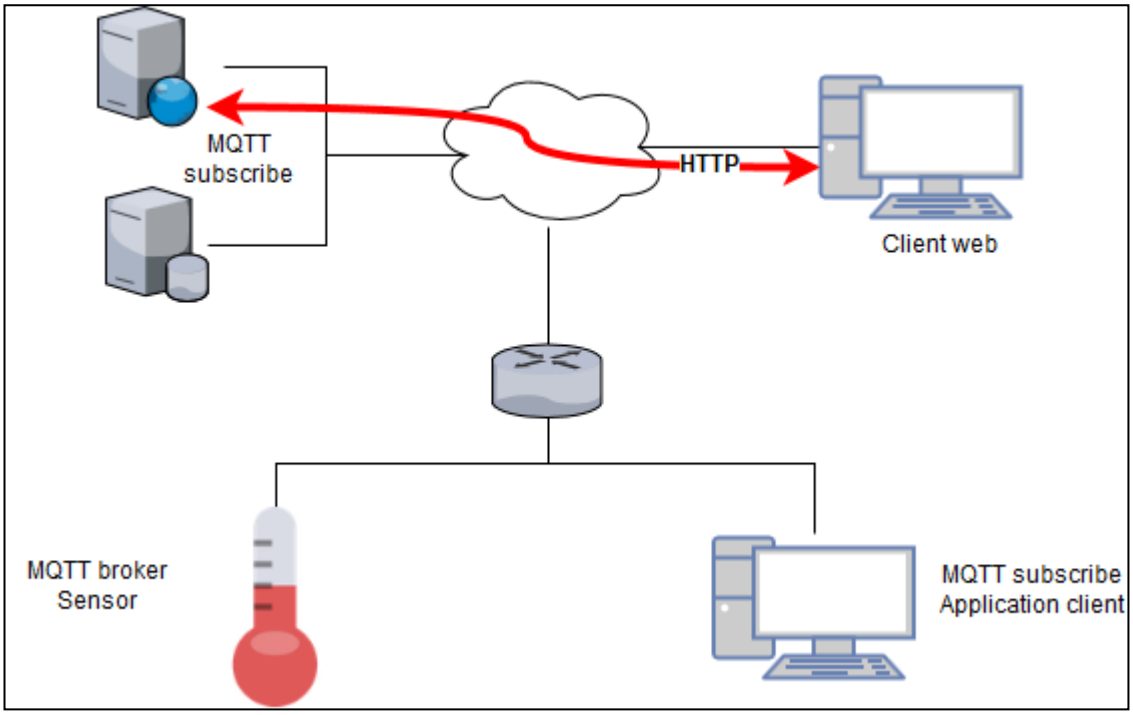
9.1.3 Node Red et MQTT

Node-Red est un outil puissant pour construire des applications de l'Internet des Objets (IoT) en mettant l'accent sur la simplification de la programmation qui se fait grâce à des blocs de code prédéfinis, appelés «nodes» pour effectuer des tâches. Il utilise une approche de programmation visuelle qui permet aux développeurs de connecter les blocs de code ensemble. Les nœuds connectés constituent un «flow». L'interface de Node-Red est accessible via un navigateur sur le port 1880. Node-Red permet de réaliser rapidement un dasboard accessible depuis un client par l'url http://<ip_serveur_node_red>:1880/ui

Afin de fournir également les données issues du capteur vers une application, il est possible d'utiliser le protocole MQTT.

MQTT est un protocole de connectivité Machine-to-Machine (M2M) de l'Internet des Objets. Il a été conçu pour le transport de messages par publication/souscription et est extrêmement léger.

Le capteur peut donc communiquer ses mesures à tous les acteurs ayant souscrit à celles-ci. Pour ce faire, le capteur doit mettre en œuvre un "broker MQTT". Les applications peuvent souscrivent au moyen d'un "client MQTT". Un serveur web associé à une BDD qui souscrit aux mesures du capteur peut donc mettre en forme une page web d'information qui sera servit aux clients HTTP.



Sur un Raspberry Pi qui tourne sur Raspbian, Node Red est déja installé. Pour le lancer, il suffit de cliquer sur le menu **Programmation** puis **Node Red**.

Node Red peut également être lancé ou stopper à partir d'un terminal :

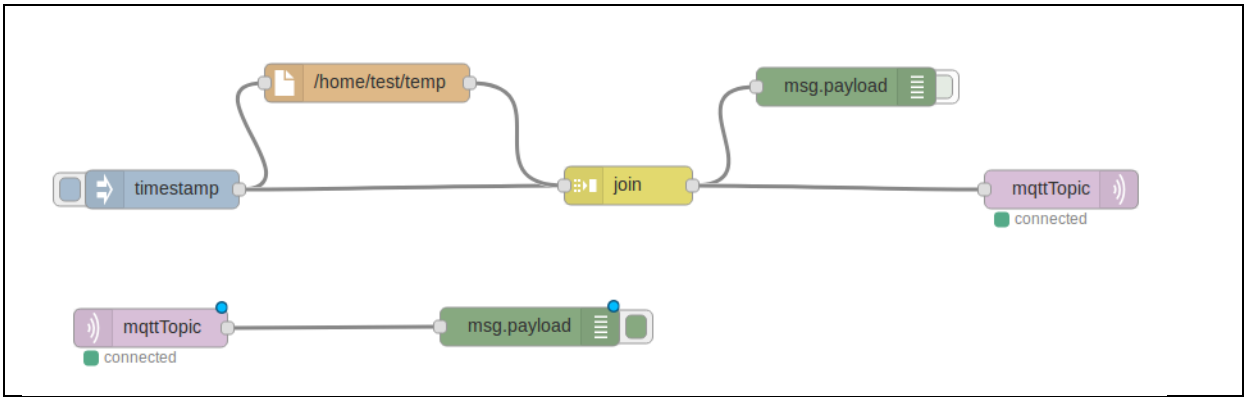
```
$ node-red-start
$ node-red-stop
```

L'accès à l'interface de programmation se fait à partir d'un navigateur. Il est préférable d'utiliser le navigateur d'un ordinateur connecté sur le même réseau que le Raspberry.

il faut également installer un broker MQTT. On en profitera pour installer également un client MQTT. On utilisera Mosquitto, un broker MQTT open source très utilisé dans les projet DIY :

```
$ sudo apt-get install mosquitto mosquitto-clients
```

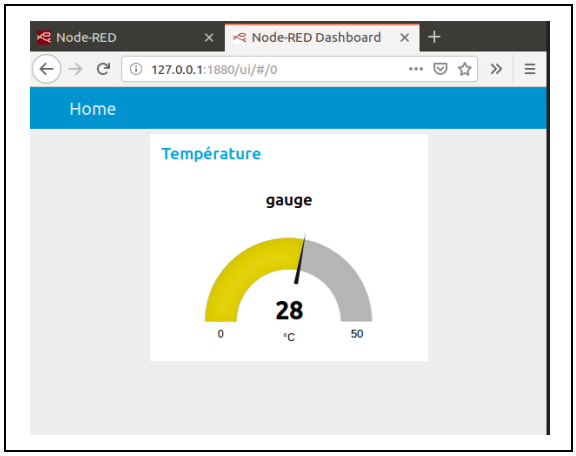
Pour compacter l'infrastructure, le serveur web (node red), le broker et le client MQTT sont tous installés sur le Raspberry. On pourra écrire la température fournie par le programme `w1` dans un fichier texte nommé `temp`, lui ajouter un timestamp représentant l'instant de la mesure et diffuser l'information via le broker MQTT.



- Le nœud **inject** injecte un timestamp sur demande ou à intervalle régulier
- Le nœud **file** obtient le contenu du fichier qui contient la dernière température fournie par le capteur
- Le nœud **join** en mode manuel combine les charges utiles de chaque message et les sépare par un caractère à spécifier (par exemple ";")
- Les nœuds **debug** permettent d'observer le résultat des opérations
- Les nœuds **mqtt** assurent la communication des données via le protocole mqtt en s'appuyant sur les services de **mosquitto**. Les paramètres de base nécessaires sont l'URL du serveur (ici **localhost**), le port (**1883** standard mqtt) et le topic (par exemple "**temp**").

Après avoir saisi le flow et configurer les nœuds, il faut le déployer pour le rendre actif.

Pour créer un dashboard, il faut compléter la palette de nœuds. Depuis le menu, accéder à Manage palette, rechercher Dashboard puis installer node-red-dashboard. On peut alors ajouter le noeud Gauge pour afficher la valeur de la température :



10 Références web

10.1 Raspberry Pi

- https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
- http://www.raspberry-projects.com/pi/category/programming-in-c
- https://learn.adafruit.com/downloads/pdf/introducing-the-raspberry-pi-2-model-b.pdf
- https://en.wikipedia.org/wiki/ARM_Cortex-A7

10.2 Crosstool-ng

- http://crosstool-ng.org/

10.3 Utilisation des GPIO

- https://www.raspberrypi.org/documentation/usage/gpio/

10.4 Utilisation du bus i2c

- https://fr.wikipedia.org/wiki/I2C
- http://www.raspberry-projects.com/pi/category/programming-in-c/i2c
- https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c

10.5 Utilisation du bus 1-wire

- http://raspberry-pi.developpez.com/cours-tutoriels/capteur/mag-pi-utiliser-port-gpio/partie-2-detection-temperature-1-wire/
- http://www.framboise314.fr/mesure-de-temperature-1-wire-ds18b20-avec-le-raspberry-pi/

10.6 Datasheet

- https://cdn-shop.adafruit.com/datasheets/TC1602A-01T.pdf
- http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

10.7 Websocketd

- http://websocketd.com/

10.8 Node red

- https://nodered.org/docs/
- http://noderedguide.com/

10.9 Mosqitto

- https://mosquitto.org/
- https://projetsdiy.fr/mosquitto-broker-mqtt-raspberry-pi/

10.10 Cours et TP des années précédentes

- http://silanus.fr/sin/?page_id=634
- https://github.com/msilanus