

IOT : PARTIE PROGRAMMATION

TP 0: I2C, Prise en main programmation du banana pi :

Comme pour les précédents Tps, devant chaque manipulation, sera annoté le temps nécessaire pour compléter la manipulation.

La somme sera égale à 3h30, vous aurez donc 30 min en cas de blocage.

L'objectif de ce TP est de se familiariser avec les différentes manières d'utiliser le banana pi en utilisant notre Gateway.

Comme vous pouvez le remarquer dans le schématique ci-dessous, la gateway est un pont entre les gpio de la carte et des composants utilisables sur ce gpio, ce qui nous évite de brancher et rebrancher à chaque TP avec des fils.

On y retrouve :

- . Une LED RGB connectée selon la couleur aux pins 7, 11 et 13

- . Un bouton poussoir connecté en pull-up à la pin 18.

- . Un connecteur femelle à 4 pins prévu pour l'utilisation d'un capteur de pression et de température BMP180.

- . 2 connecteurs femelles pins, un prévu pour connecté un module radio NRF au banna pi et l'autre prévu pour connecté un autre module radio NRF à la carte arduino .

- . Une carte arduino Pro mini dont la Pin 12 est elle-même connectée à une LED.

Lorsqu'on programme sur une carte Raspberry ou Banana etc., plusieurs nomenclatures sont utilisées pour les PINS.

Elles sont numérotées de manières physiques sur le schématique, mais à chaque Pin correspond aussi un nom de type GPIO X et PAX ou PCX.

Les codes et les bibliothèques utilisées sur internet faites de bases pour des Raspberry Pi doivent donc toujours être adaptées à notre banana Pi.

Ci-dessous le PIN numbering du banana pi qu'il faut utiliser lorsque vous prendrez en main celui-ci.

3V3 Power	1	2	5V Power
GPIO2 SDA1 I2C	3	4	5V Power
GPIO3 SCL1 I2C	5	6	Ground
GPIO4	7	8	GPIO14 UART0_TXD
Ground	9	10	GPIO15 UART0_RXD
GPIO17	11	12	GPIO18
GPIO27	13	14	Ground
GPIO22	15	16	GPIO23
3V3 Power	17	18	GPIO24
GPIO10 SPI0_MOSI	19	20	Ground
GPIO9 SPI0_MISO	21	22	GPIO25
GPIO11 SPI0_SCLK	23	24	GPIO5 SPI0_CS0_N
Ground	25	26	GPIO7 SPI0_CS1_N
ID_SD IO ID COPY04	27	28	ID_SC IO ID COPY04
GPIO5	29	30	Ground
GPIO6	31	32	GPIO12
GPIO13	33	34	Ground
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
Ground	39	40	GPIO21

GPIO Pin	Name	Default Function	Function2 : GPIO	Function3
CON2-P01	VCC-3V3			
CON2-P02	VCC-5V			
CON2-P03	GPIO0-SDA	PA12-EINT12		
CON2-P04	VCC-5V			
CON2-P05	GPIO0-SCK	PA11-EINT11		
CON2-P06	GND			
CON2-P07	PWM1	PA6-EINT6		
CON2-P08	UART3-TX	PA13-EINT13	SPI1-CS	
CON2-P09	GND			
CON2-P10	UART3-RX	PA14-EINT14	SPI1-CLK	
CON2-P11	UART2-RX	PA1-EINT1		
CON2-P12	UART3-CTS	PA16-EINT16	SPI1-MISO	
CON2-P13	UART2-TX	PA0-EINT0		
CON2-P14	GND			
CON2-P15	UART2-CTS	PA3-EINT3		
CON2-P16	UART3-RTS	PA15-EINT15	SPI1-MOSI	
CON2-P17	VCC-3V3			
CON2-P18	PC4	PC4		
CON2-P19	SPI0-MOSI	PC0		
CON2-P20	GND			
CON2-P21	SPI0-MISO	PC1		
CON2-P22	UART2-RTS	PA2-EINT2		
CON2-P23	SPI0-CLK	PC2		
CON2-P24	SPI0-CS	PC3		
CON2-P25	GND			
CON2-P26	PC7	PC7		

Bien, maintenant que vous avez en tête sur quels éléments vous allez travailler, nous allons passer à utiliser notre Gateway et notre carte.

Avant d'accéder aux GPIOs, on va accéder de deux manières différentes à la LED intégrée au banana Pi (voyant rouge) lorsque vous allumez le banana PI ;

Il y a deux manières de gérer la led, on peut le faire soit via un fichier exécutable .sh que l'on va créer, soit via un programme C.

Dans un premier lieu, nous allons essayer de faire fonctionner celle-ci à l'aide d'un fichier exécutable :

nano bash_LED.sh

Ensuite on copie colle le code ci-dessous et on enregistre :

```
#!/bin/bash
LED3_PATH=/sys/class/leds/bananapi-m2-plus:red:pwr
function removeTrigger
{
echo "none" >> "$LED3_PATH/trigger"
}
echo "Starting the LED Bash Script"
if [ $# != 1 ]; then
echo "There is an incorrect number of arguments. Usage is:"
echo -e " bashLED Command \n where command is one of "
echo -e " on, off, flash or status \n e.g. bashLED on "
exit 2
fi
echo "The LED Command that was passed is: $1"
if [ "$1" == "on" ]; then
echo "Turning the LED on"
removeTrigger
echo "1" >> "$LED3_PATH/brightness"
elif [ "$1" == "off" ]; then
echo "Turning the LED off"
removeTrigger
echo "0" >> "$LED3_PATH/brightness"
fi
```

On Remarque que le chemin ou est repertorié la led dans le noyau est indiqué par

LED3_PATH=/sys/class/leds/bananapi-m2-plus:red:pwr

```
GNU nano 2.7.4 File: bash_LED.sh

#!/bin/bash
LED3_PATH=/sys/class/leds/bananapi-m2-plus:red:pwr
function removeTrigger
{
echo "none" >> "$LED3_PATH/trigger"
}
echo "Starting the LED Bash Script"
if [ $# != 1 ]; then
echo "There is an incorrect number of arguments. Usage is
echo -e " bashLED Command \n where command is one of "
echo -e " on, off, flash or status \n e.g. bashLED on "
exit 2
fi
echo "The LED Command that was passed is: $1"
if [ "$1" == "on" ]; then
echo "Turning the LED on"
removeTrigger
echo "1" >> "$LED3_PATH/brightness"
elif [ "$1" == "off" ]; then
```

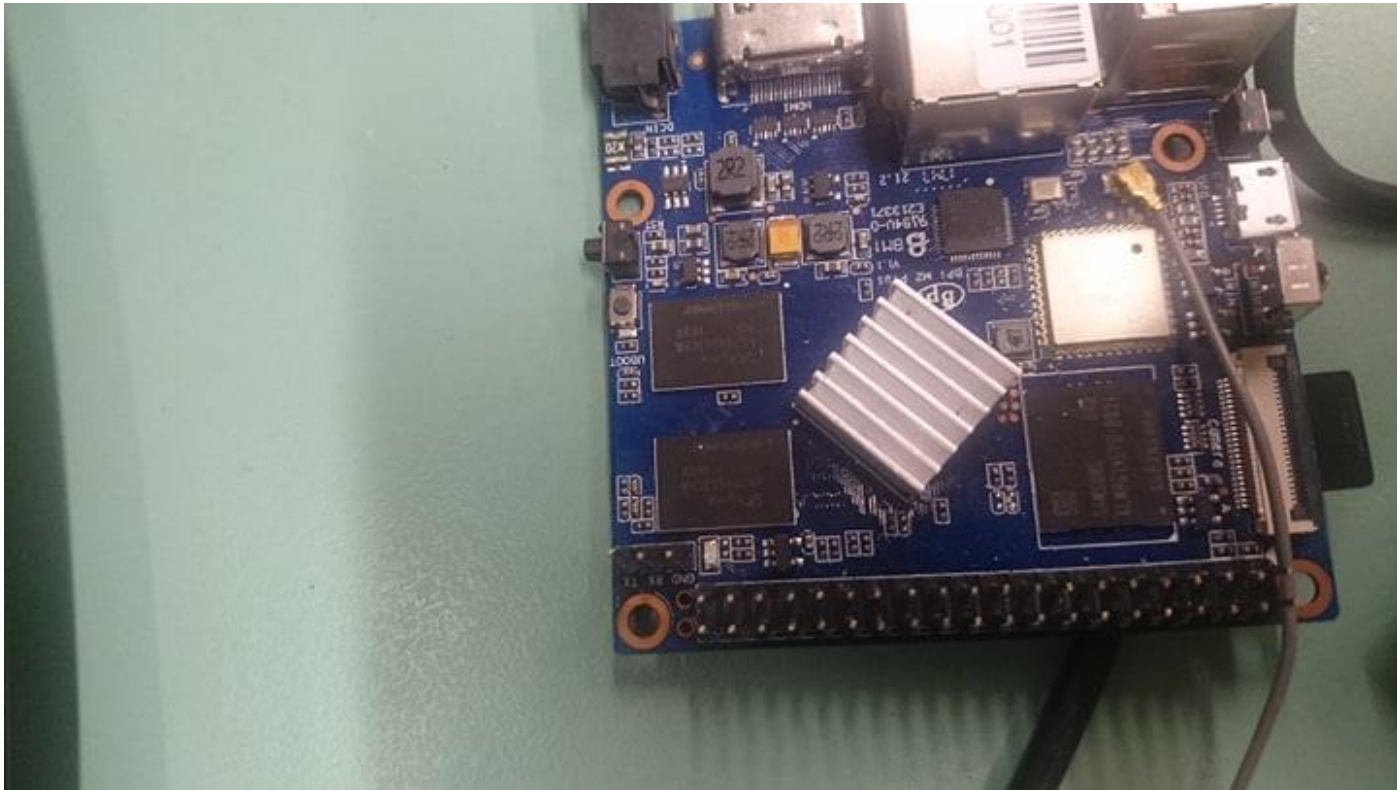
Il y a d'abord une fonction removeTrigger, qui affiche none et qui associe cet affichage au chemin de la LED3/trigger, ensuite dans le programme principal, on affiche le démarrage du script, et s'il y a un nombre d'arguments différents de 1, n affiche une erreur expliquant la façon de passer la commande.

Dans le cas contraire si l'argument est « on », on allume la led, et si l'argument est « off » on l'éteint.

On essaye de passer la commande avec un off pour éteindre la led et tester la commande :

```
bash bash_LED.sh off
```

```
root@yhstg:~# bash bash_LED.sh off
Starting the LED Bash Script
The LED Command that was passed is: off
Turning the LED off
root@yhstg:~#
```



La led s'est bien éteinte.

On peut rendre le fichier exécutable et tester la commande pour allumer la led :

```
chmod +x bash_LED.sh  
sudo ./bash_LED.sh on
```

```
root@yhstg:~# chmod +x bash_LED.sh  
root@yhstg:~# sudo ./bash_LED.sh on  
Starting the LED Bash Script  
The LED Command that was passed is: on  
Turning the LED on
```

Normalement la LED devrait s'être allumée.

Le même programme en ouvrant le fichier où se trouve la LED en langage C cette fois peut être fait :

<https://gist.github.com/Mizaystom/9da4fb627c940231e3e54e223bf442f9>

Exercice : créer un fichier, coller le programme et le compiler.

La compilation se fait avec gcc :

```
gcc fichier.c -o fichier
```

Nb : si gcc ne fonctionne pas : télécharger la librairie :

```
apt-get install build-essential
```

Une fois compiler vous pouvez l'exécuter comme précédemment avec un :

```
sudo ./fichier on
```

Vous devez alors avoir le même fonctionnement que précédemment.

Test des GPIO :

Comme vu au début du TP, Le port GPIO est un connecteur à 40 points qui nous permet d'accéder aux ports d'entrée sorties du processeur.

Autrement dit, s'ils ne fonctionnent pas, on ne pourra pas utiliser tous les modules externes nécessitant une connexion au raspberry.

Nous allons donc les tester.

Pour ainsi faire, nous allons tout d'abord télécharger la bibliothèque Wiring Pi, qui permet de relier les pins du gpio au registre de la carte banana pi, ce qui nous permet de ne pas avoir à chercher manuellement les adresses afin d'utiliser nos ports.

```
git clone git://git.drogon.net/wiringPi
```

```
cd wiringPi
```

```
./build
```

Ensuite lancer un gpio readall :

```
geii@bpstage:~$ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode |   Name   | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |   3.3v   |      |   | 1 | 2 |     |   5v     |     |     | |
| 12 |  8 | SDA.0    | ALT5 | 0 | 3 | 4 |     |   5V     |     |     |
| 11 |  9 | SCL.0    | ALT5 | 0 | 5 | 6 |     |   0v     |     |     |
|  6 |  7 | GPIO.7    | OUT  | 1 | 7 | 8 | 0 | ALT4 | TxD3    | 15 | 13 |
|     |     |   0v     |      |   | 9 | 10 | 0 | ALT4 | RxD3    | 16 | 14 |
|  1 |  0 | Rx/D2    | ALT3 | 0 | 11 | 12 | 0 | ALT3 | GPIO.1  |  1 | 110 |
|  0 |  2 | TxD2     | ALT3 | 0 | 13 | 14 |   |     |   0v     |     |     |
|  3 |  3 | CTS2     | ALT3 | 0 | 15 | 16 | 1 | IN  | GPIO.4  |  4 | 68 |
|     |     |   3.3v   |      |   | 17 | 18 | 1 | IN  | GPIO.5  |  5 | 71 |
| 64 | 12 | MOSI     | ALT4 | 0 | 19 | 20 |   |     |   0v     |     |     |
| 65 | 13 | MISO     | ALT4 | 0 | 21 | 22 | 0 | ALT3 | RTS2    |  6 |  2 |
| 66 | 14 | SCLK     | ALT4 | 0 | 23 | 24 | 0 | ALT4 | CE0     | 10 | 67 |
|     |     |   0v     |      |   | 25 | 26 | 0 | ALT3 | GPIO.11 | 11 | 21 |
| 19 | 30 | SDA.1    | ALT4 | 0 | 27 | 28 | 0 | ALT4 | SCL.1   | 31 | 18 |
|  7 | 21 | GPIO.21  | ALT3 | 0 | 29 | 30 |   |     |   0v     |     |     |
|  8 | 22 | GPIO.22  | ALT3 | 0 | 31 | 32 | 0 | ALT5 | RTS1    | 26 | 200 |
|  9 | 23 | GPIO.23  | ALT3 | 0 | 33 | 34 |   |     |   0v     |     |     |
| 10 | 24 | GPIO.24  | ALT3 | 0 | 35 | 36 | 0 | ALT5 | CTS1    | 27 | 201 |
| 20 | 25 | GPIO.25  | ALT3 | 0 | 37 | 38 | 0 | ALT5 | TxD1    | 28 | 198 |
|     |     |   0v     |      |   | 39 | 40 | 0 | ALT5 | RxD1    | 29 | 199 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode |   Name   | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode |   Name   | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```


Vous remarquez qu'à côté de chaque pin, un numéro wPi est attribué. Ce qui veut dire que lorsqu'on utilise la bibliothèque WiringPi, et qu'on veut définir une Led par exemple, c'est ce numéro que nous devons attribuer.

Ci-dessous un schéma de la carte banana pi avec le numéro Wpi correspondant à chaque PIN. Seules les Pins utilisées par la Gateway sont numérotées.



Exercice : à l'aide d'un programme en C et de la bibliothèque WiringPi Pi, faire clignoter la led RGB en bleu. Consultez le schématique plus haut.

Correction :

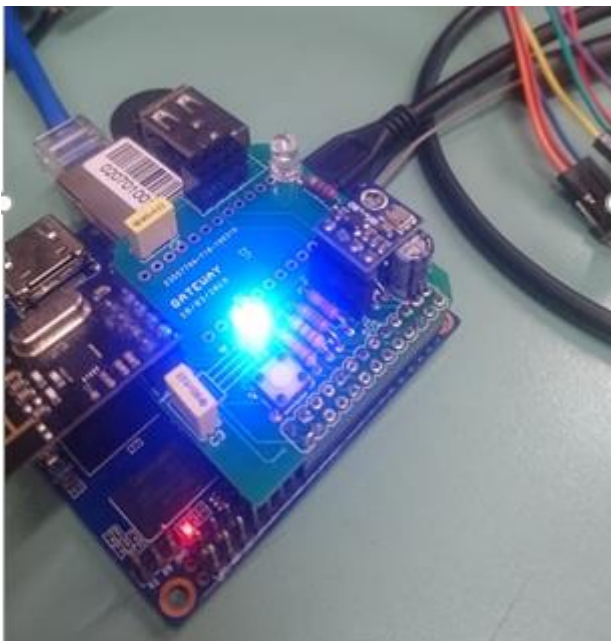
```
#include <wiringPi.h>

int main (void)
{
    wiringPiSetup();
    pinMode(0,OUTPUT);
    for(;;)
    {
        digitalWrite(0,HIGH) ; delay(500);
        digitalWrite(0,LOW) ; delay(500);
    }
    return 0;
}
```

Compilation :

```
gcc fichier.c -o fichier -lwiringPi
```

lwiringPi est une bibliothèque externe, il faut donc l'associer lors de la compilation.



Vous pouvez tester de faire fonctionner toutes les couleurs de la LED RGB en changeant le programme.

Ctrl + z pour arrêter le programme après l'avoir lancer avec sudo ./fichier.

Exercice : modifier le programme pour allumer la led quand le bouton est appuyé, et l'éteindre quand il ne l'est pas.

Le bouton est connecté à la pin 4 de la bibliothèque Wpi.

Maintenant que vous avez pris en main les gpio en utilisant la bibliothèque WiringPi, nous allons manipuler ceux-ci directement avec les fichiers du système (sysfs).

Les numéros des broches que l'on va utiliser sont celles sous la forme PAX et PCX.

Lorsqu'on veut utiliser une broche PA : le numéro à définir est le nombre devant la broche en question.

Si c'est une broche PC, le numéro à définir est 64+le nombre devant PC.

Un exemple serait plus clair.

Supposons qu'on veuille allumer la led en rouge. On regarde dans le schématique, celle-ci est connectée à la pin physique 13. On regarde le numéro de broche qui lui est correspondant dans le tableau donné en début de TP.

Dans notre cas, c'est PA0.

On se met en super utilisateur :

```
sudo su
```

Ensuite : on tape la commande suivant pour envoyer 0, et exporter la broche que l'on veut utiliser.

```
echo 0 > /sys/class/gpio/export
```

On met ensuite la broche que l'on vient d'exporter en out car on veut allumer et éteindre la led.

```
echo out > /sys/class/gpio/gpio0/direction
```

La led est à cathode commune elle devrait déjà être allumée à 0.

Ensuite, envoyé 1 en valeur pour éteindre la led :

```
echo 1 > /sys/class/gpio/gpio0/value
```

Exercice : éteindre et allumer la led dans les deux autres couleurs, ensuite suivant le même principe et en entrée, essayer de lire l'état du bouton poussoir avec la commande more (PC4).

Correction bouton poussoir :

Exportation : PC4 donc pin 64+4

```
echo 68 > /sys/class/gpio/export
```

Direction : (on veut lire)

```
echo in > /sys/class/gpio/gpio68/direction
```

Se mettre dans le répertoire du fichier :

```
cd /sys/class/gpio/gpio68
```

Lire :

```
more value
```

Le bouton est en résistance de pull-up .

Ce qui veut dire que lorsqu'on appui sur le bouton, on est à 0v, et lorsqu'on appui pas on est à 3V3.

Donc lorsque vous n'appuyez pas, more renverra 1, et lorsque vous appuierez more renverra 0.

```
root@bpstage:/sys/class/gpio/gpio68# more value
1
```

Nous allons mettre en place un script.sh pour tester les leds :

```
sudo nano testrgb.sh
```

Le code est donné ci-dessous, cependant il faudra l'adapter à notre LED RGB et des erreurs s'y seront glissées. Vous devez le corriger. Prenez soin de comprendre les commentaires. La correction est à la page suivante. Pour mettre exécutable :

```
chmod +testrgb.sh
```

```

#!/bin/bash

# Description : turn on / turn off every second GPIO 0, 4 68

# root ? // On test si on est bien en root
function isRoot {
if [[ $EUID -ne 0 ]]; then
    echo "you must be root -> use sudo test_leds.sh " 1>&2
    exit 1
fi
}

# Init // fonction d'initialisation de toutes les leds
function init_GPIO {
    for i in 0 4 68
    do
        echo out > /sys/class/gpio/gpio$i/direction
    done
}

# if gpio doesn't exist, create // si l'utilisation a échoué et que gpio n'existe pas, exporter
function create_GPIO {
    for i in 0 4 68
    do
        if [ -e /sys/class/gpio/gpio$i ]; then
            echo $i > /sys/class/gpio/export
        fi
    done
}

# Turn on // allumage
function on {
    for i in 0 4 68
    do
        /bin/echo 1 > /sys/class/gpio/gpio$i/value
    done
}

# Turn off // eteindre
function off {
    for i in 0 4 68
    do
        /bin/echo 0 > /sys/class/gpio/gpio$i/value
    done
}

# le script // on appelle les fonctions
isRoot
create_GPIO
init_GPIO
while :
do
on
sleep 1
off
sleep 1
done

```

```

#!/bin/bash

# Description : turn on / turn off every second GPIO 6, 1 0
# root ?
function isRoot {
if [[ $EUID -ne 0 ]]; then

    echo "you must be root -> use sudo test_leds.sh " 1>&2

    exit 1
fi
}
# Init

function init_GPIO {
for i in 0 1 6
do
echo out > /sys/class/gpio/gpio$i/direction
done
}

# if gpio doesn't exist, create
function create_GPIO {
for i in 0 1 6
do
if [ -e /sys/class/gpio/gpio$i ]; then
echo $i > /sys/class/gpio/export
fi
done
}

# Turn on
function on {
for i in 0 1 6
do
/bin/echo 0 > /sys/class/gpio/gpio$i/value
done
}

# Turn off
function off {
for i in 0 1 6
do
/bin/echo 1 > /sys/class/gpio/gpio$i/value
done
}

# le script
isRoot
create_GPIO
init_GPIO
while :
do
on
sleep 1
off
sleep 1
done

```

Maintenant que vous avez vu comment prendre le contrôle des gpio avec sysfs et avec la bibliothèque WiringPi, en utilisant des programmes en C compilés avec gcc, nous allons voir une autre méthode de compilation qui est le makefile.

Les Makefiles sont des fichiers, généralement appelés makefile ou Makefile, utilisés par le programme make pour exécuter un ensemble d'actions, comme la compilation d'un projet, l'archivage de document, la mise à jour de site, etc. Cet article présentera le fonctionnement de makefile au travers de la compilation d'un petit projet en C.

Nous allons voir rapidement comment ça fonctionne mais je vous conseille vivement de consulter ce site :

<https://gl.developpez.com/tutoriel/outil/makefile/>

Nous allons donc comme sur le site, créer un dossier où l'on va créer 3 fichiers :

```
mkdir testmakefile
```

```
cd testmakefile
```

Allez sur le site web et créer les 3 fichiers demandés à l'aide de nano en copiant les codes.

```
geii@bpstage:~$ cd testmakefile/
geii@bpstage:~/testmakefile$ sudo nano hello.c
[sudo] password for geii:
geii@bpstage:~/testmakefile$ sudo nano hello.h
geii@bpstage:~/testmakefile$ sudo nano main.c
geii@bpstage:~/testmakefile$ ls
hello.c hello.h main.c
geii@bpstage:~/testmakefile$
```

Un Makefile est un fichier constitué de plusieurs règles de la forme :

Sélectionnez

cible: dependance
commandes

Chaque commande est précédée d'une tabulation.

Lors de l'utilisation d'un tel fichier via la commande make la première règle rencontrée, ou la règle dont le nom est spécifié, est évaluée. L'évaluation d'une règle se fait en plusieurs étapes :

- Les dépendances sont analysées, si une dépendance est la cible d'une autre règle du Makefile, cette règle est à son tour évaluée.

- Lorsque l'ensemble des dépendances est analysé et si la cible ne correspond pas à un fichier existant ou si un fichier dépendance est plus récent que la règle, les différentes commandes sont exécutées.

Premier makefile de test :

```
sudo nano makefile
```

et copier le code ci-dessous :

```
hello: hello.o main.o
    gcc -o hello hello.o main.o

hello.o: hello.c
    gcc -o hello.o -c hello.c -W -Wall -ansi -pedantic

main.o: main.c hello.h
    gcc -o main.o -c main.c -W -Wall -ansi -pedantic
```

Vous pouvez tester votre makefile :

```
make  
sudo ./hello
```

Regardons de plus près sur cet exemple comment fonctionne un Makefile :

Nous cherchons à créer le fichier exécutable hello, la première dépendance est la cible d'une des règles de notre Makefile, nous évaluons donc cette règle. Comme aucune dépendance de hello.o n'est une règle, aucune autre règle n'est à évaluer pour compléter celle-ci.

Deux cas se présentent ici : soit le fichier hello.c est plus récent que le fichier hello.o, la commande est alors exécutée et hello.o est construit, soit hello.o est plus récent que hello.c est la commande n'est pas exécutée. L'évaluation de la règle hello.o est terminée.

Les autres dépendances de hello sont examinées de la même manière puis, si nécessaire, la commande de la règle hello est exécutée et hello est construit.

Plusieurs cas ne sont pas gérés dans l'exemple précédent :

- Un tel Makefile ne permet pas de générer plusieurs exécutables distincts.
- Les fichiers intermédiaires restent sur le disque dur même lors de la mise en production.
- Il n'est pas possible de forcer la régénération intégrale du projet

Ces différents cas conduisent à l'écriture de règles complémentaires :

- all : généralement la première du fichier, elle regroupe dans ces dépendances l'ensemble des exécutables à produire.
- clean : elle permet de supprimer tout les fichiers intermédiaires.
- mrproper : elle supprime tout ce qui peut être régénéré et permet une reconstruction complète du projet.

Il faudrait donc rajouter des règles supplémentaires pour paufinner notre makefile.

Par exemple si on tape make clean, ça ne parcherait pas car on a pas rajouter la règle suivant à notre make file.

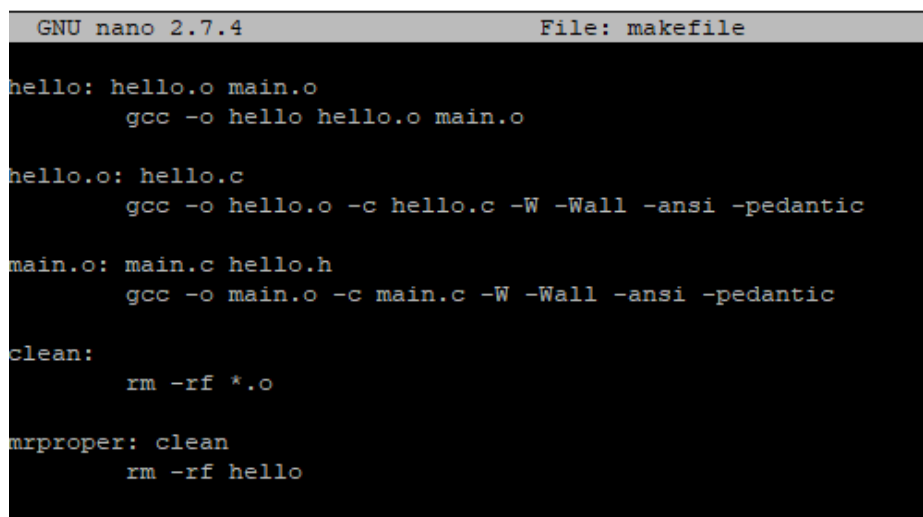
```
clean:
    rm -rf *.o

mrproper: clean
    rm -rf hello
```

Modifier votre makefile précédent pour rajouter la règle clean.

Ensuite refaire un make, lancer votre programme et faire un makeclean pour voir si votre make efface les fichiers.

Résultat attendu :



```
GNU nano 2.7.4 File: makefile

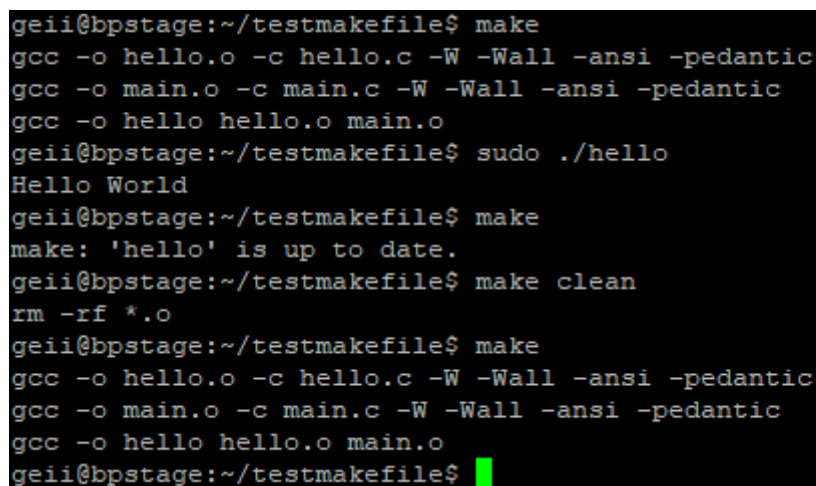
hello: hello.o main.o
    gcc -o hello hello.o main.o

hello.o: hello.c
    gcc -o hello.o -c hello.c -W -Wall -ansi -pedantic

main.o: main.c hello.h
    gcc -o main.o -c main.c -W -Wall -ansi -pedantic

clean:
    rm -rf *.o

mrproper: clean
    rm -rf hello
```



```
geii@bpstage:~/testmakefile$ make
gcc -o hello.o -c hello.c -W -Wall -ansi -pedantic
gcc -o main.o -c main.c -W -Wall -ansi -pedantic
gcc -o hello hello.o main.o
geii@bpstage:~/testmakefile$ sudo ./hello
Hello World
geii@bpstage:~/testmakefile$ make
make: 'hello' is up to date.
geii@bpstage:~/testmakefile$ make clean
rm -rf *.o
geii@bpstage:~/testmakefile$ make
gcc -o hello.o -c hello.c -W -Wall -ansi -pedantic
gcc -o main.o -c main.c -W -Wall -ansi -pedantic
gcc -o hello hello.o main.o
geii@bpstage:~/testmakefile$
```

Visual Studio Community

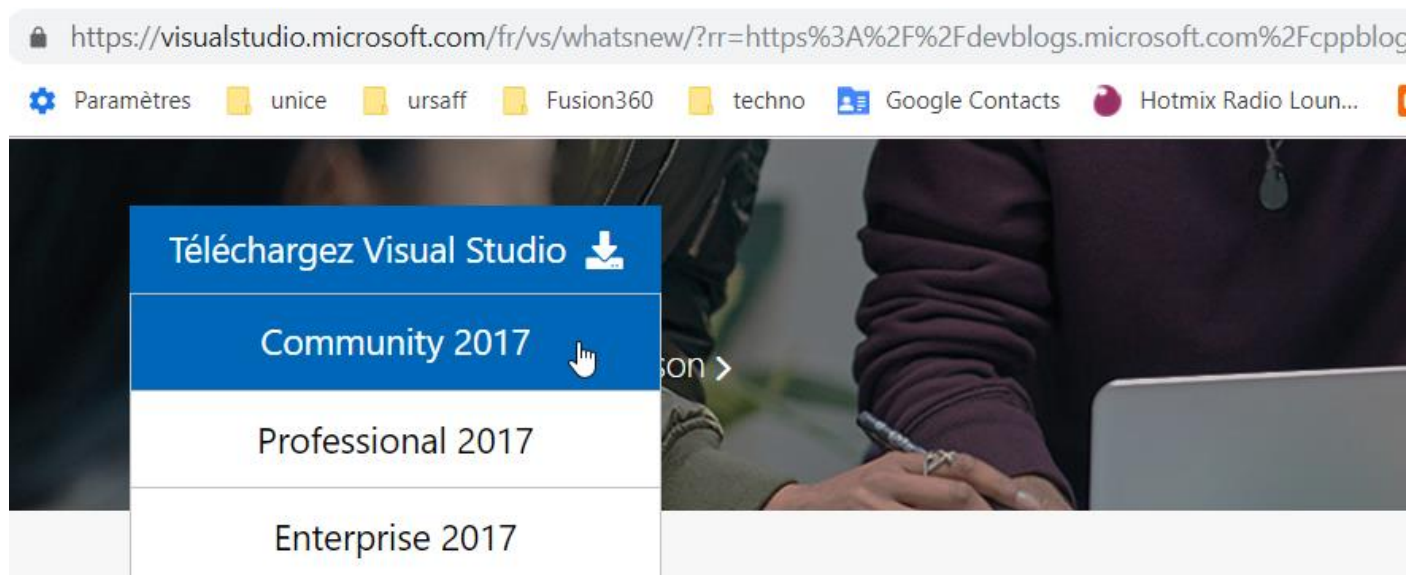
Si vous n'êtes pas à l'aise avec nano, plusieurs autres solutions s'offrent à vous.

Vous pouvez, taper vos codes sur votre IDE préféré et les déplacer avec Winscp (voir tp administration).

Vous pouvez utiliser un serveur vnc, pour avoir une interface graphique sur votre banana pi et télécharger code blocks (cf tp administration).

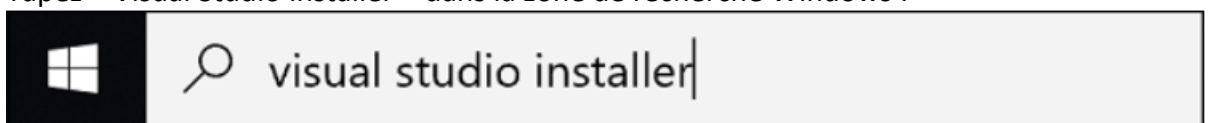
Ou bien vous pouvez utiliser le logiciel communautaire Visual studio Community.

Dans un premier lieu, télécharger Visual Studio Community si ce n'est pas déjà fait :



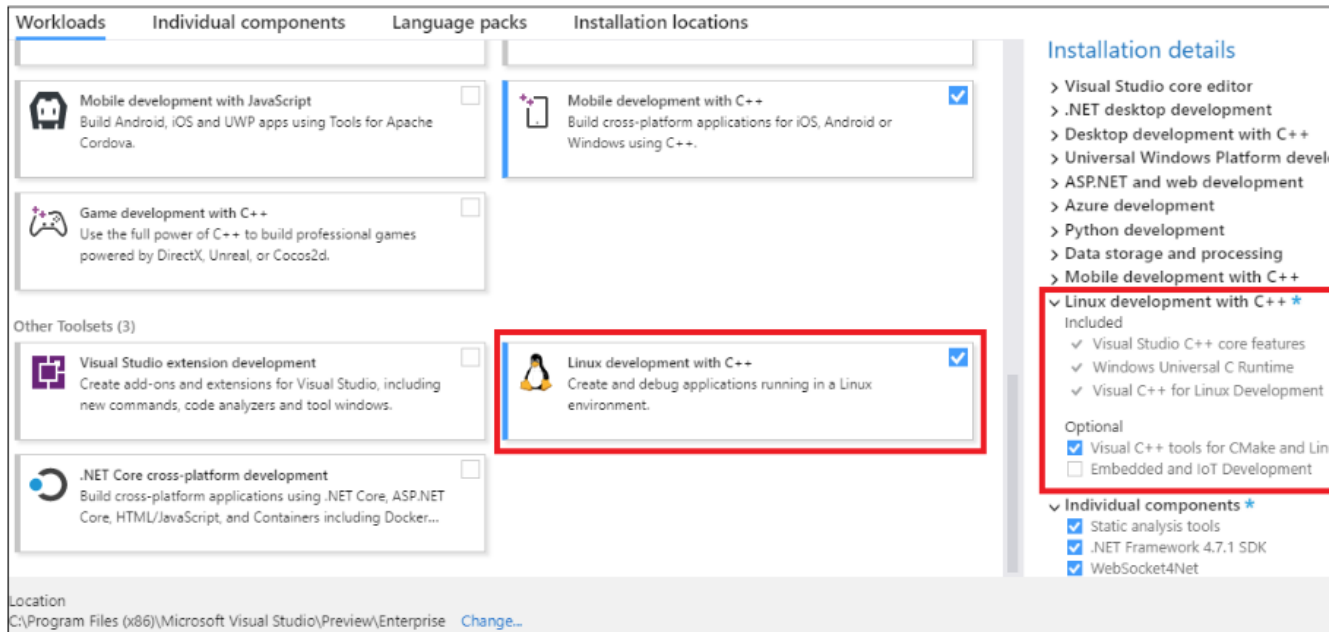
Ensuite,

1. Tapez « Visual Studio Installer » dans la zone de recherche Windows :



2. Recherchez le programme d'installation dans les résultats situés sous **Applications**, puis double-cliquez dessus. Quand le programme d'installation s'ouvre, choisissez **Modifier**, puis cliquez sur l'onglet **Charges de travail**. Faites défiler vers le bas jusqu'à **Autres ensembles d'outils** et sélectionnez la charge de

travail Développement Linux en C++.











3. Si vous utilisez CMake ou que vous ciblez des plateformes incorporées ou IoT, accédez au volet **Détails de l'installation** à droite, sous **Développement Linux en C++**, développez **Composants facultatifs** et choisissez les composants dont vous avez besoin.
4. Cliquez sur **Modifier** pour continuer l'installation.

Une fois le logiciel ouvert, créer un projet :

Visual Studio 2019

Ouvrir les éléments récents

 Meteo.sln	2018-10-08 1:53 PM
C:\Users\Yacine\source\repos\Meteo	
 App10.sln	2018-09-24 4:55 PM
C:\Users\Yacine\source\repos\App10	
 App11.sln	2018-09-24 4:54 PM
C:\Users\Yacine\source\repos\App11	
 App9.sln	2018-09-24 4:20 PM
C:\Users\Yacine\source\repos\App9	
 App8.sln	2018-09-24 4:19 PM
C:\Users\Yacine\source\repos\App8	
 App7.sln	2018-09-24 3:58 PM
C:\Users\Yacine\source\repos\App7	
 App6.sln	2018-09-24 3:55 PM
C:\Users\Yacine\source\repos\App6	
 App5.sln	2018-09-24 3:51 PM
C:\Users\Yacine\source\repos\App5	

Prise en main



Cloner ou extraire du code

Obtenir du code à partir d'un dépôt en ligne, par exemple GitHub ou Azure DevOps



Ouvrir un projet ou une solution

Ouvrir un projet ou un fichier .sln Visual Studio local



Ouvrir un dossier local

Naviguer parmi du code et le modifier dans n'importe quel dossier



Créer un projet

Choisir un modèle de projet avec génération de modèles automatique de code pour bien démarrer

[Continuer sans code →](#)

Créer un projet


Modèles de projet récents

La liste des modèles récemment consultés sera affichée ici.


Langage

Plateforme


Type de projet

 **Projet d'éléments partagés**
Un projet d'éléments partagés permet de partager des fichiers entre plusieurs projets.


C++WindowsAndroidIOSLinuxBureauConsoleBibliothèqueUWPJeuxMobile

 **Application console**
Exécutez du code dans une fenêtre de terminal Linux. Permet d'afficher "Hello" par défaut.


C++LinuxConsole

 **Projet vide**
Démarez à partir de zéro en C++ pour Linux. Aucun fichier de départ n'est fourni.

C++LinuxConsole

 **Clignotement sur Raspberry Pi**
Application de clignotement de LED à l'aide de WiringPi pour Raspberry Pi.

C++LinuxIoTConsole

 **Projet Makefile**
Utilisez votre propre système de compilation pour compiler en C++ pour Linux.

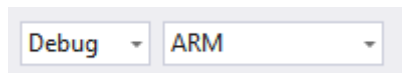
C++LinuxBibliothèqueConsoleBureau

Ma nouvelle solution

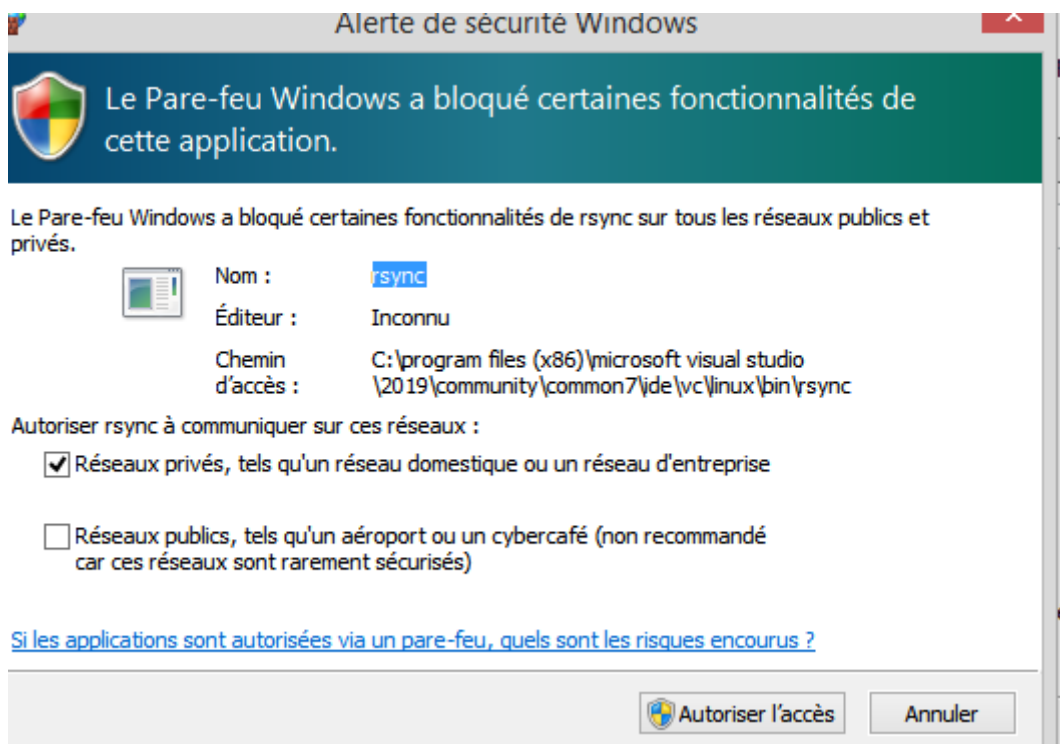
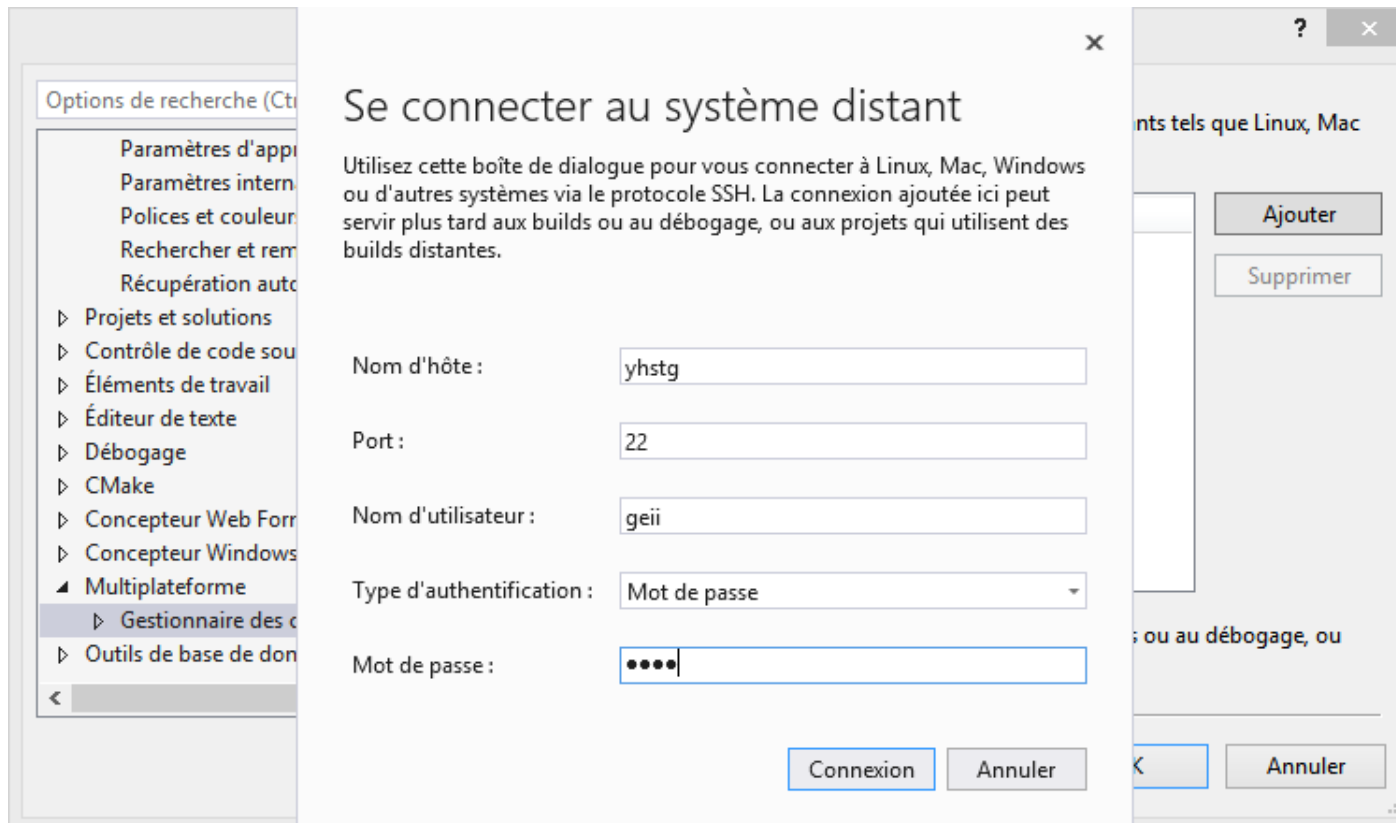
Retour

Suivant

Choisir clignotement sur Raspberry Pi, une fois le projet créé, assurez vous qu'on est en ARM et pas en ARM64 car notre processeur est un processeur 32 bits.

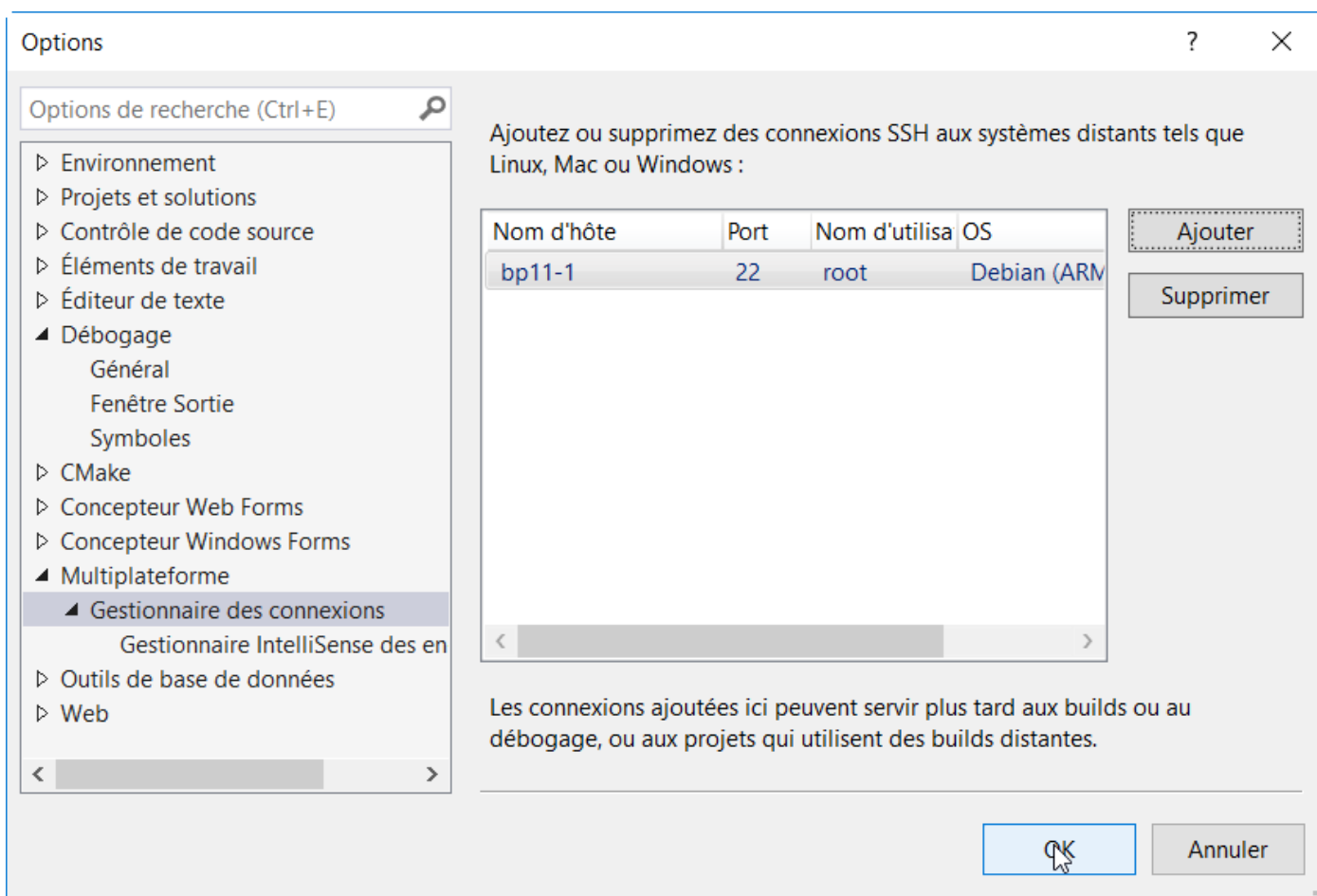
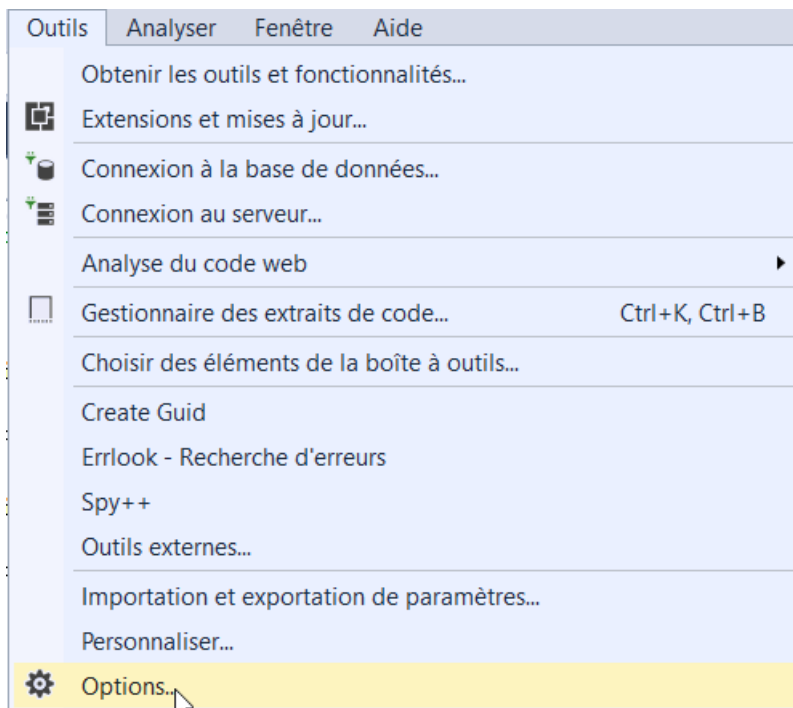


Ensuite dans outils → options → multiplateforme → gestionnaire de connexions

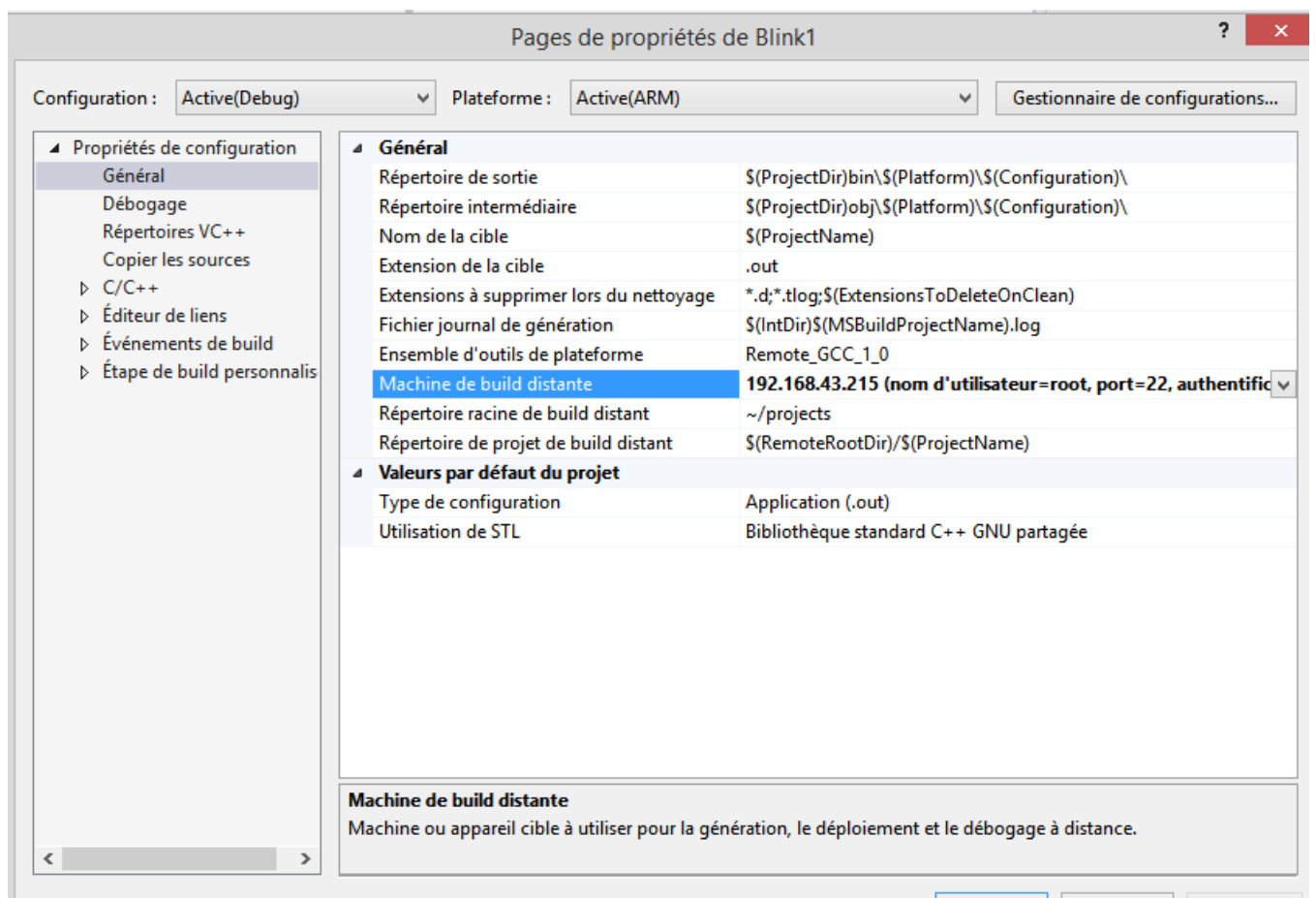
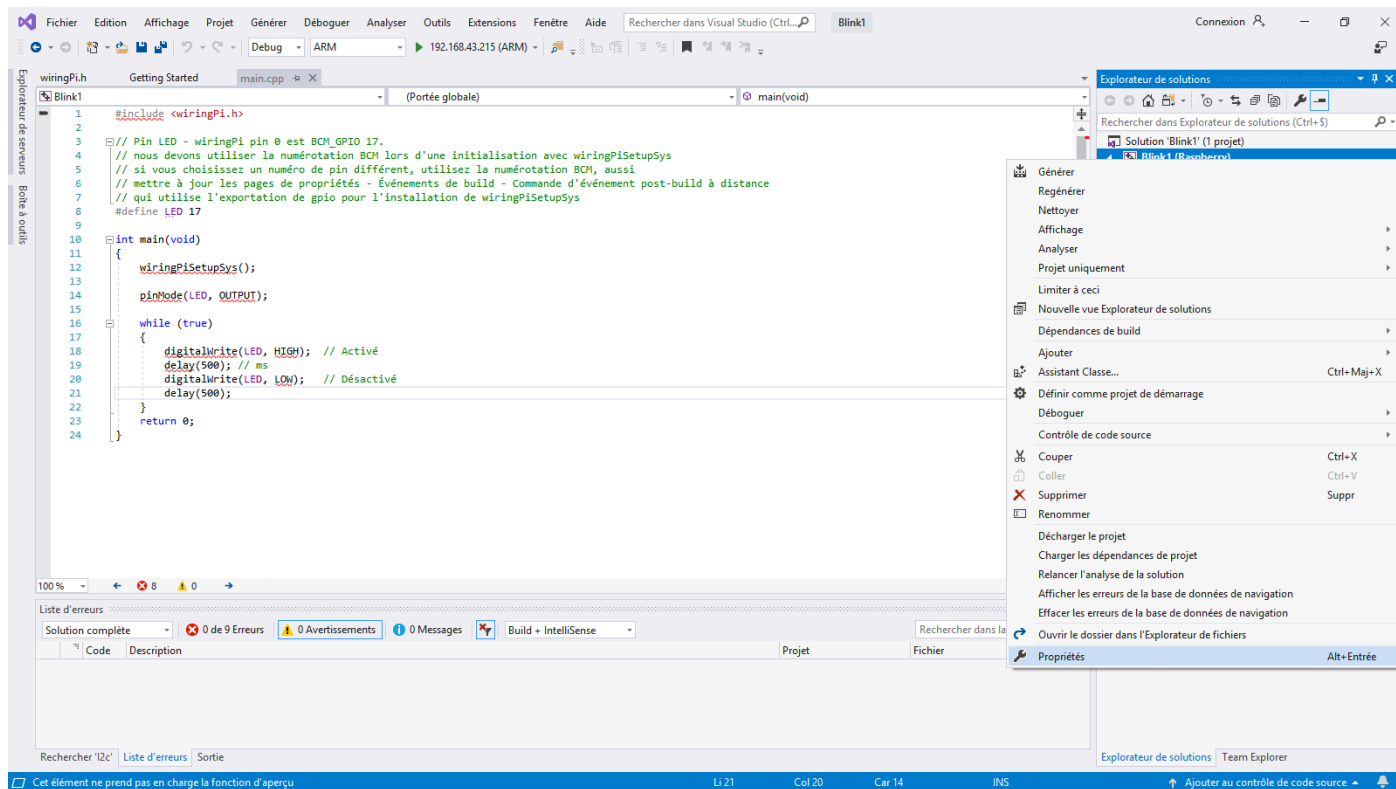


Autoriser les accès avec votre pare-feu.

Vous pouvez ensuite appuyer sur ok, dans ce menu après avoir sélectionner la cible sur laquelle vous voulez vous connecter :



Si vous avez deux noms, un avec root et l'autre avec geii créer, vous pouvez changer en allant



Ensuite, une fois tout ceci configuré, sur votre console Linux (putty par exemple) installer les librairies suivantes :

```
sudo apt-get install openssh-server g++ gdb gdbserver
```

Ensuite dans votre main, adapter le programme pour allumer l'une des Leds Rouge bleu ou verte, rajouter la bibliothèque stdio pour afficher des messages sur le terminal de VS community ou 2017.

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>

#define LED 0 //ou celle que vous voulez 0, c'est
bleu

int main(void)
{
    if (wiringPiSetup() == -1) //setup la bibliothèque
wiringpi, si il y a echec, on sort
    {
        exit(1);
    }

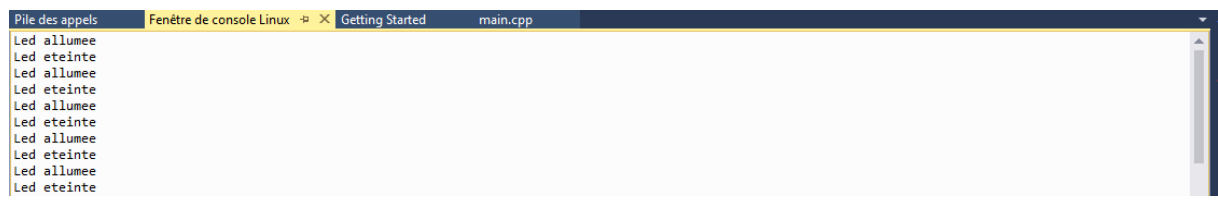
    pinMode(LED, OUTPUT);

    while (true)
    {
        digitalWrite(LED, 0); // Activé
        printf("Led allumee \r");
        delay(500); // ms
        digitalWrite(LED, 1); // Désactivé
        printf("Led eteinte \r");
        delay(500);
    }
    return 0;
}
```

Compilez :

▶ 10.23.30.3 (ARM) ▾

Votre Led devrait clignoter et des messages sur le terminal devraient afficher ceci :



```
Pile des appels  Fenêtre de console Linux -R X Getting Started main.cpp
Led allumee
Led eteinte
Led allumee
Led eteinte
Led allumee
Led eteinte
Led allumee
Led eteinte
Led allumee
Led eteinte
Led allumee
Led eteinte
```