

Développement à base de cartes de prototypage rapide MBED

Programmation réseau avec MBED

Prérequis : langage C, programmation objet, notion de programmation socket : client, serveur, réseau IP...



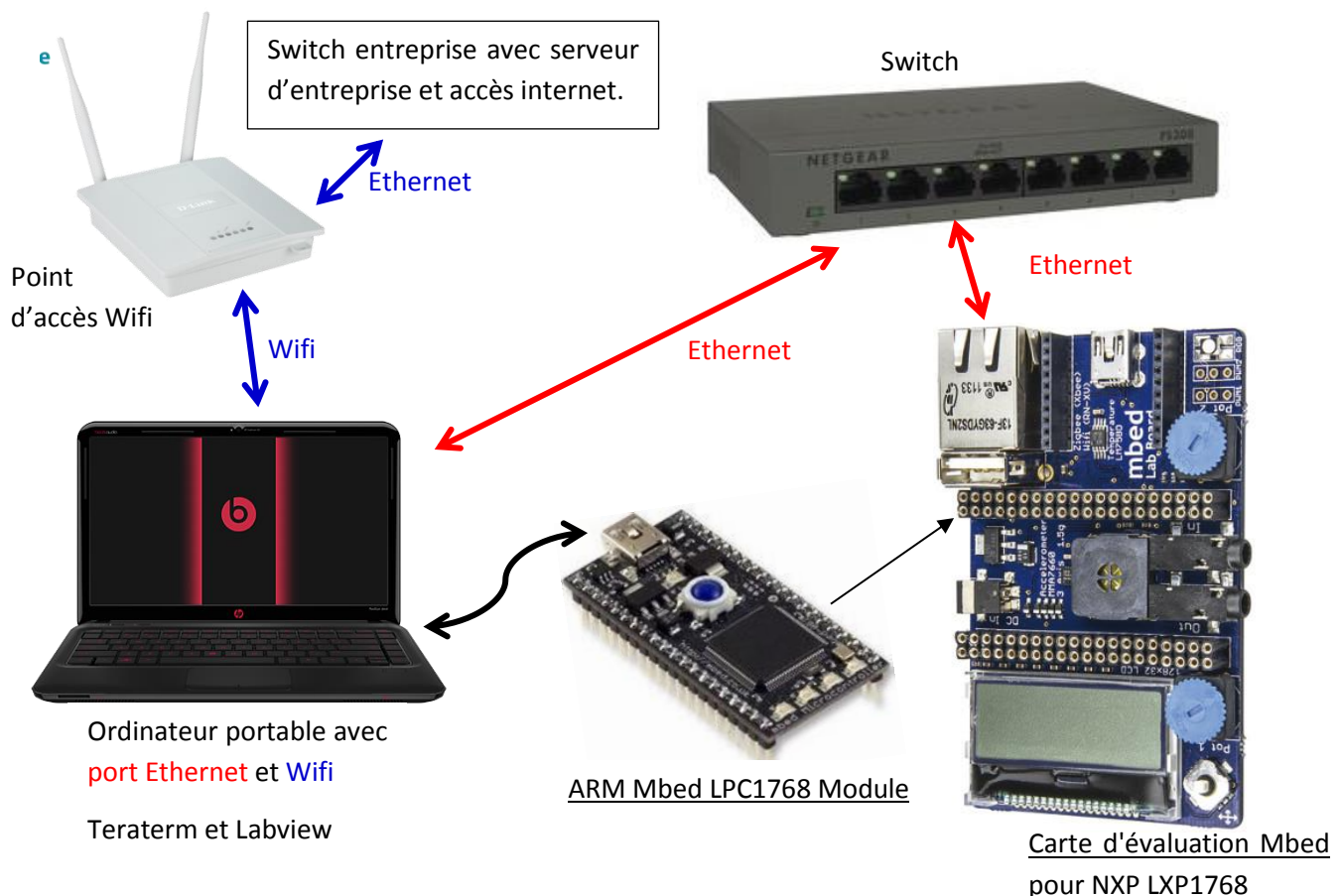
Table des matières

Ce que vous saurez faire.....	3
Matériel proposé.....	3
Remarque importante pratique.....	4
Il faudra donc s'adapter pour le cas de l'utilisation de 2 Pc.....	4
Création du projet.....	4
Mise en place d'un port série virtuel sur la connexion USB.....	4
Création d'un nouveau projet et préparation de l'environnement MBED pour effectuer de la programmation reseau.....	6
Programmation reseau – mise en situation et définition des rôles.....	8
Paramètres Ip et Tcp.....	9
Sauvegarde du projet en cours.....	9
Nous continuons à travailler avec le projet « TP4_prog-reseau ».....	9
Mise en place de la couche IP.....	10
Programmation reseau premier serveur.....	12
Programmation reseau deuxième serveur.....	14
Programmation reseau troisième serveur : station météo communicante.....	17
Programmation reseau client : capteur température.....	22

Ce que vous saurez faire.

Vous allez apprendre à effectuer de la programmation réseau à partir d'une solution de prototypage MBED. Vous serez capable de dialoguer en socket entre un PC et une carte MBED, dans un premier temps en mode serveur puis en mode client. Vous saurez aussi utiliser la liaison USB de ma carte MBED en mode série virtuelle.

Matériel proposé.



Un seul ordinateur est utilisé en attribuant :

- la connexion Wifi pour la partie utilisation du réseau : internet donnant accès notamment au logiciel de développement MBED et aux ressources du réseau local.
- la connexion Ethernet pour la partie développement (programmation réseau avec la carte MBED). Le switch et les 2 câbles Ethernet (rouge) peuvent être remplacés par un câble croisé.
- Il est possible d'utiliser 2 ordinateurs ou 1 ordinateur avec 2 cartes réseau.

L'utilisation d'une seule interface réseau est possible mais non conseillée dans le cas d'un réseau partagé avec plusieurs personnes. Dans un réseau professionnel cette organisation séparant la partie développement réseau de la partie utilisation est obligatoire avant d'éviter de mettre en dysfonctionnement le réseau d'entreprise en cas d'erreur de programmation. Pour le cas d'un réseau sous Scribe, il est conseillé d'utiliser 2 ordinateurs.

L'ordinateur est relié comme vous le savez en USB à la carte MBED pour la programmation. Le port USB peut être utilisé en port série virtuel notamment pour envoyer des informations de débogage.

Remarque importante pratique.

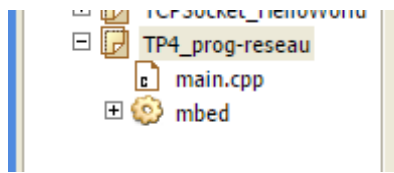
Ce document est écrit pour une utilisation avec un seul Pc et 2 interfaces réseaux. Nous aurons dans ce cas :

- 1 PC attribué à la connexion Wifi pour la partie utilisation du réseau.
- 1 PC attribué à la connexion Ethernet pour la partie développement.

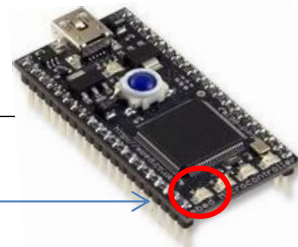
Il faudra donc s'adapter pour le cas de l'utilisation de 2 Pc.

Création du projet.

Dans votre interface de travail sur mbed.org, vous créez un nouveau projet TP4_prog-reseau. Ce projet vous servira de base pour la programmation réseau.



Nous utilisons la template « Blinky LED hello world »



Nous testons le fonctionnement, la LED indiquée clignote.

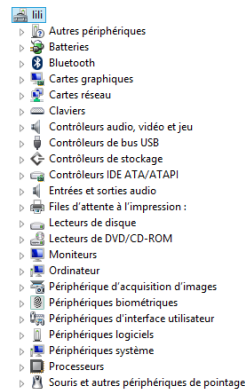
Mise en place d'un port série virtuel sur la connexion USB.

Nous allons mettre en place un port série virtuel sur la liaison USB. Ce port série virtuel sera utile pour tous vos développements et vous permettra de mettre un dialogue entre le pc de développement et la carte MBED. Vous pourrez dans vos programmes ajouter des messages de fonctionnement et ainsi voir comment votre programme se déroule.

Pour Windows, il suffit d'installer le driver pour le port série virtuel pour la carte MBED :

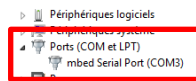
<http://developer.mbed.org/handbook/Windows-serial-configuration>

Après avoir installé le port COM virtuel, nous allons le tester :



Les périphériques de Windows ne possèdent pas de port série MBED, lorsque que le driver n'est pas installé.

Lorsque le driver est installé et que vous branchez la carte MBED un port série apparait. Attention si vous utilisez un PC pour lequel vous avez des droits limités (par exemple Scribe) vous devez faire installer le driver par l'administrateur du poste.



Pour utiliser un port série virtuel dans les programmes, il suffit d'instancier un objet avec la classe serial dont les attributs sont USBTX, USBRX :

```
Serial pc(USBTX, USBRX);
```

Le fichier main.cpp devient.


```
#include "mbed.h"
DigitalOut myled(LED1);
Serial pc(USBTX, USBRX);
int main() {
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

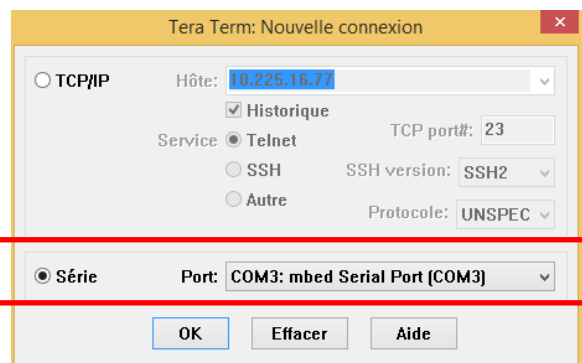
Le port série pc possède les attributs de la classe Serial. Il aura donc les mêmes attributs et méthodes qu'un port série physique.

Nous pourrions donc utiliser la méthode printf() pour envoyer un message ou les méthodes readable(), getch() pour lire un caractère ...

Le fichier main.cpp pour l'envoi du message "Hello, I'm happy my Virtual Serial port is ok." est donné ci-dessous :

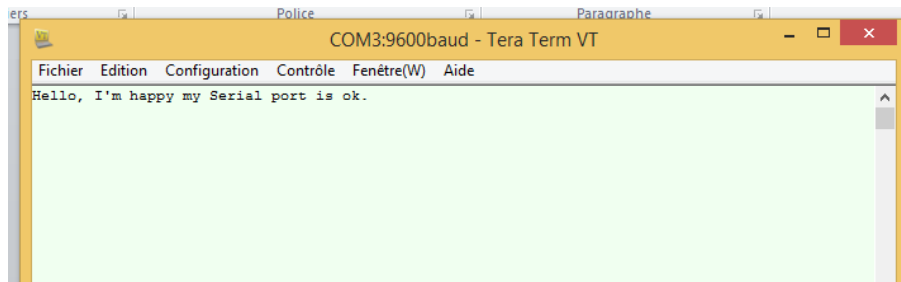
```
#include "mbed.h"
DigitalOut myled(LED1);
Serial pc(USBTX, USBRX);
int main() {
    pc.printf("Hello, I'm happy my Virtual Serial port is ok.\n\r");
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

Il ne reste plus qu'à tester son fonctionnement à l'aide du logiciel TeraTerm . Personnellement j'utilise la version portable. Vous lancez Teraterm et sélectionnez le port virtuel série :



Vous compilez le programme, vous le chargez dans la mémoire de la carte MBED.

Vous appuyez sur le bouton RESET de la carte MBED pour exécuter le programme et en principe vous obtenez le résultat suivant :



Tout va bien, nous allons pouvoir commencer la programmation réseau.

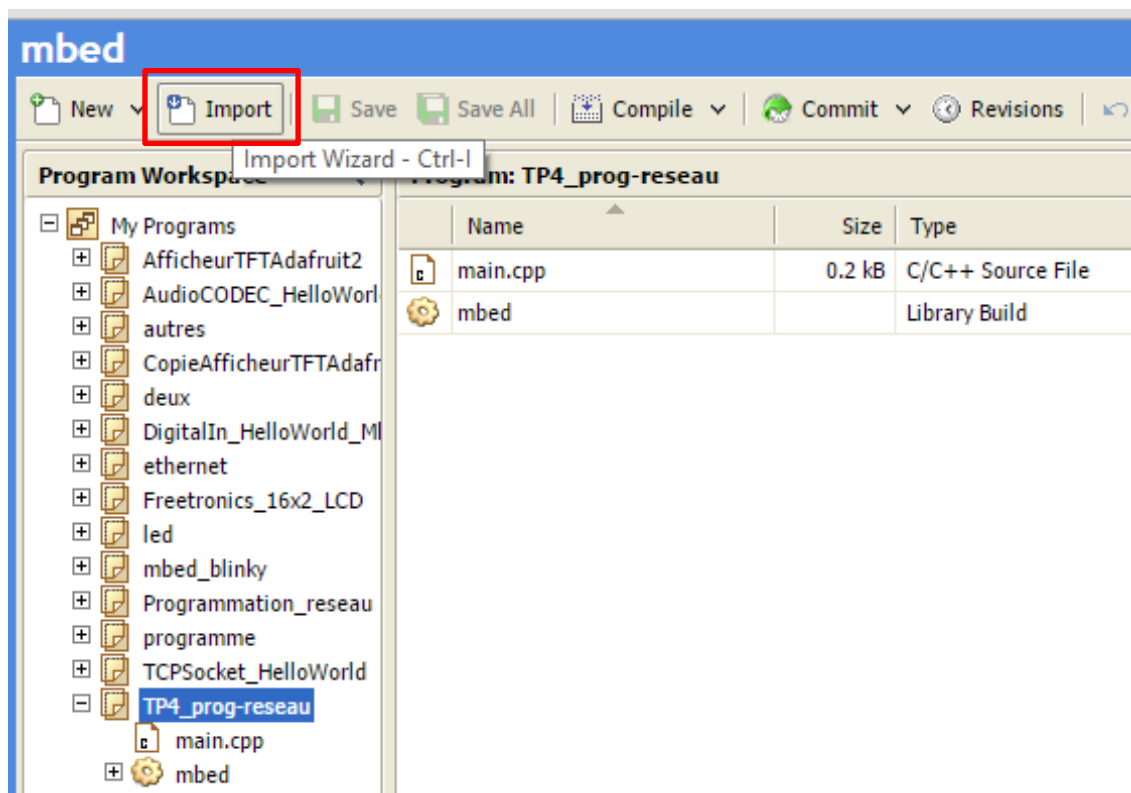
Le port virtuel va servir à envoyer des messages.

Création d'un nouveau projet et préparation de l'environnement MBED pour effectuer de la programmation réseau.

Nous allons maintenant ajouter au projet courant les bibliothèques pour la programmation réseau.

- **Ethernet Interface** : bibliothèque officielle pour la gestion des sockets, [voir](#).
- **MBED-RTOS** : ajoute des fonctionnalités temps réel au système d'exploitation MBED. Elle est nécessaire au fonctionnement de la bibliothèque **Ethernet Interface**.

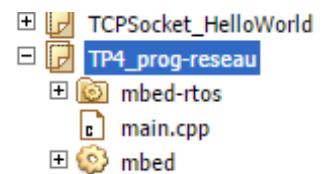
Vous ajoutez dans un premier temps la bibliothèque MBED-RTOS à notre projet, pour cela vous importez la bibliothèque RTOS en cliquant sur « Import ».



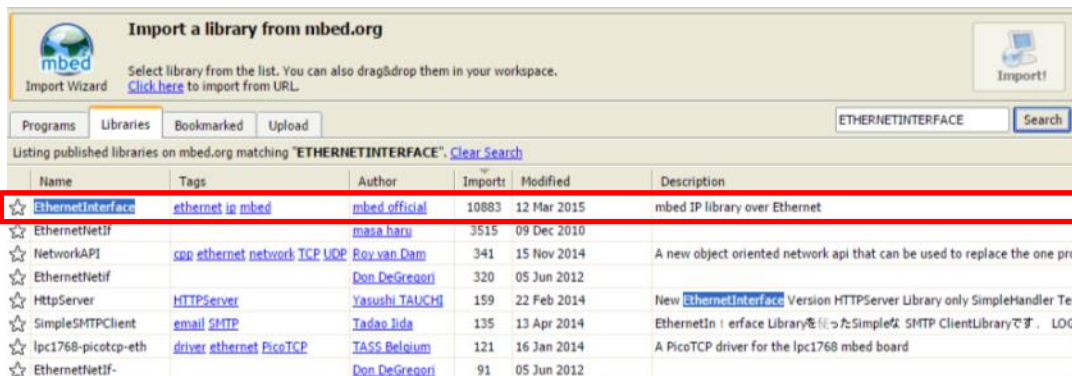
Vous recherchez la chaine MBED-RTOS



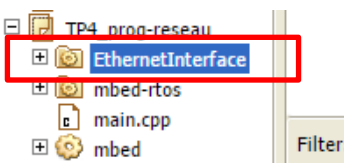
Vous sélectionnez la librairie « MBED officiel ». L'importation s'effectue. Vous pouvez vérifier que la librairie a été ajoutée à votre projet.



Maintenant vous ajoutez la librairie « Ethernet Interface » à votre projet. Vous Procédez de même que précédemment.



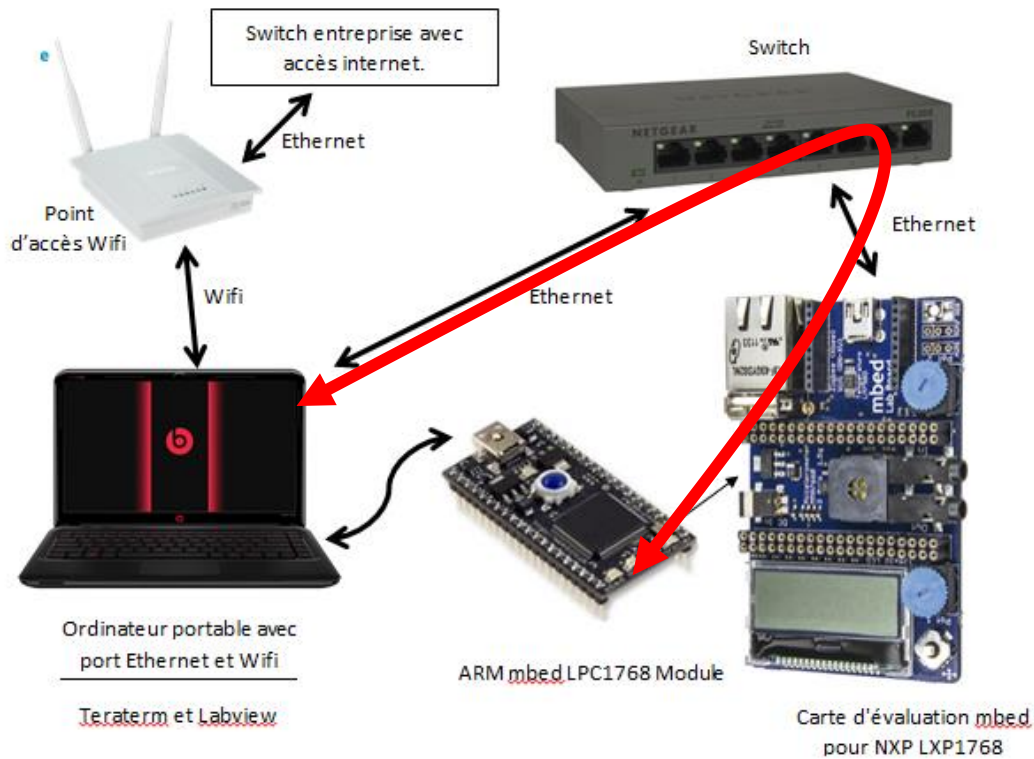
Importer la librairie, voici le résultat.



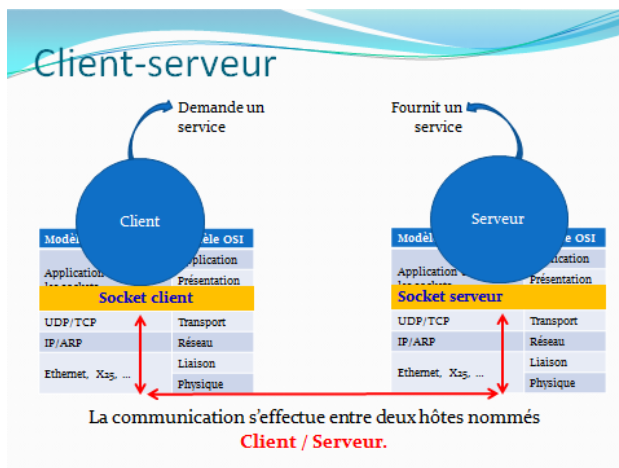
Votre environnement est prêt pour le développement réseau.

Programmation réseau – mise en situation et définition des rôles.

Nous allons maintenant établir une communication entre la carte MBED et le pc à travers le réseau :



Remarque : il n'existe pas de serveur DHCP au niveau du réseau, où se situe la carte MBED. Il faudra donc utiliser un adressage IP fixe.



La programmation réseau permet d'établir un dialogue en mode client-serveur :

Nous allons choisir le protocole connecté donc TCP.

Le PC aura le rôle de client et utilisera pour cela le client Teraterm.

La carte MBED aura le rôle de serveur. Nous allons la programmer pour cela.

Paramètres Ip et Tcp

Nous allons définir les paramètres Ip, qui seront utilisés tout au long de cette activité.

Configuration Ip

Adresse réseau	192.168.1.0
Masque sous réseau	255.255.255.0
Passerelle	192.168.1.254
PC	192.168.1.1
Carte MBED	192.168.1.10

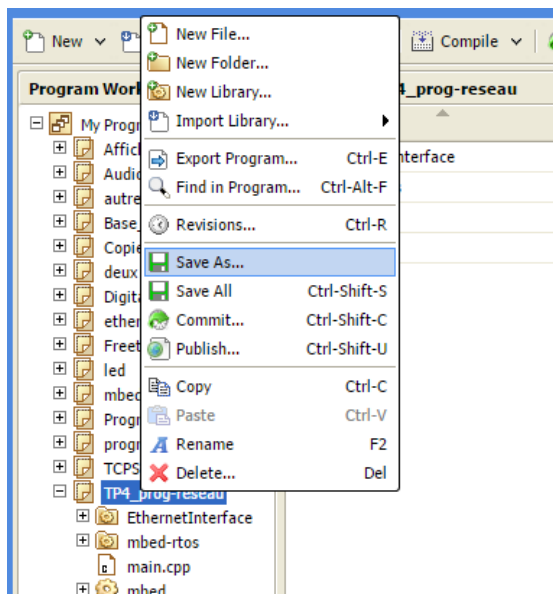
Attention l'adressage réseau coté Ethernet doit être différent de celui coté Wifi.

Configuration Tcp

Port utilisé	2000
--------------	------

Sauvegarde du projet en cours.

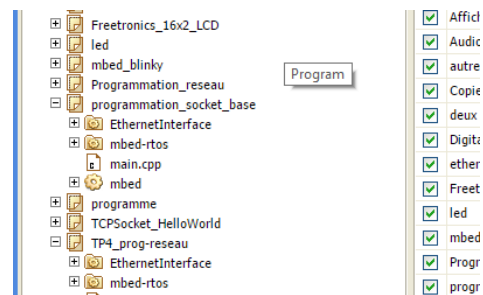
Le projet courant « TP4_prog-reseau » est une base intéressante pour la programmation réseau.



Il est possible de garder une copie avant de continuer.

Vous indiquez le nouveau nom par exemple « programmation_socket_base » et validez.

Le nouveau projet apparaît :



Nous continuons à travailler avec le projet « TP4_prog-reseau ».

Mise en place de la couche IP.

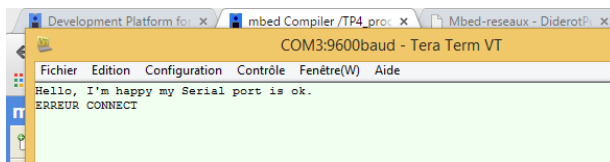
Nous allons utiliser le port série virtuel pour afficher les informations de fonctionnement.

Au niveau du programme, il va s'agir d'ajouter un objet de class « EthernetInterface » et ensuite de vérifier son fonctionnement. Nous allons ajouter les éléments en rouge au programme précédent.

```
#include "mbed.h"
#include "EthernetInterface.h"
EthernetInterface eth;
DigitalOut myled(LED1);
Serial pc(USBTX, USBRX);
int main() {
    pc.printf("Hello, I'm happy my Serial port is ok.\n\r");
    if (eth.init()!=0) {
        pc.printf("ERREUR INIT ETHERNET\r\n");
        return -1;
    }
    if (eth.connect()!=0) {
        pc.printf("ERREUR CONNECT\r\n");
        return -1;
    }
    pc.printf("L adresse Ip est %s\r\n", eth.getIPAddress());
    pc.printf("IP est OK, vous pouvez continuer\r\n ");
    ...
}
```

Les méthodes `init()` et `connect()` lorsqu'elles se déroulent sans erreur renvoient une valeur nulle. En cas d'erreur d'une de ces méthodes, nous constatons qu'un message est envoyé vers l'interface série virtuel et que le programme est quitté en envoyant la valeur -1 au système d'exploitation MBED.

Nous testons le programme, le résultat est le suivant :



Nous constatons que la fonction `connect()` ne s'est pas bien déroulée.

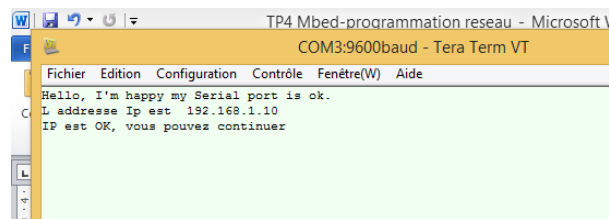
Nous voyons l'utilité d'utiliser le port virtuel pour envoyer des messages.

L'erreur est normale, nous avons oublié qu'il n'y a pas de serveur DHCP présent dans le réseau. L'adresse IP de la carte MBED n'est pas définie. Il faut attribuer les paramètres IP au moment de l'initialisation de l'interface Eth.

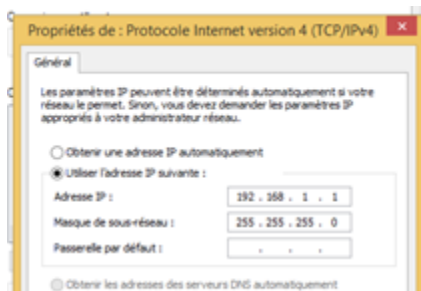
Nous ajoutons les éléments suivants :

- Après l'instanciation de Eth :
 - EthernetInterface eth;
 - char ip[] = "192.168.1.10";
 - char masque[] = "255.255.255.0";
 - char passerelle[] = "192.168.1.255";
- Au niveau de init() :
 - if (eth.init(ip,masque,passerelle)!=0) {...

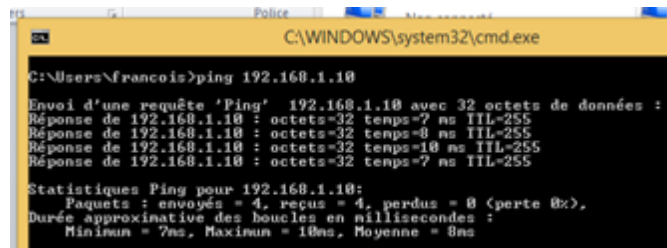
Nous recommençons le test. Le résultat est maintenant correct.



Nous configurons le paramétrage IP du PC.



Nous pouvons vérifier que la carte MBED répond à un ping :



Programmation réseau premier serveur.

Nous allons dans un premier temps sauvegarder le projet « TP4_prog-reseau » sous le nom « TP4_prog-reseau-premier-serveur ». Nous utiliserons ce projet dans cette partie.

Ce premier serveur n'a pas de véritable utilité mais il servira de base pour le développement de serveur plus complexe. Son fonctionnement :

Serveur : carte MBED

Création socket serveur

Affectation d'un port d'écoute au serveur

Préparation du serveur pour l'écoute

Création socket client

Attente connexion d'un client

Envoi message au client

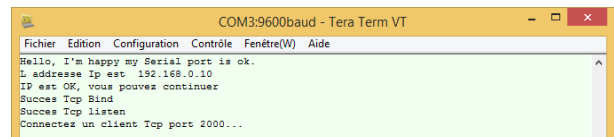
Fermeture connexion client

```
#define serveur_port 2000
TCPSocketServer server;
server.bind (serveur_port);
server.listen();
TCPSocketConnection client;
server.accept (client)
client.send("Hello\r\n",strlen("Hello\r\n"));
client.close();
```

Dans la phase développement, il est conseillé d'envoyer des messages à travers l'interface série virtuelle, afin de vérifier le fonctionnement.

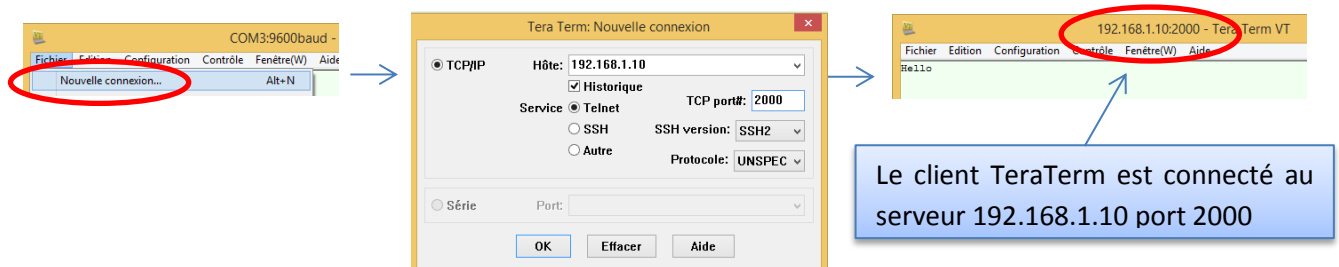
Vous trouverez le programme du serveur page suivante. Le programme n'est pas commenté, il faut le faire au fur et à mesure de l'écriture. Nous allons vérifier son fonctionnement. Vous compilez le programme et le copiez dans la carte MBED.

Vous connectez avec Teraterm le port série virtuel, si cela n'est pas fait avant de « reseter » la carte MBED. Tout se passe bien vous obtenez le message suivant :



Nous constatons que la LED ne clignote pas, cela veut dire que la fonction « server.accept (client) » est bloquante, elle attend la connexion d'un client.

Vous créez une nouvelle connexion, qui va ouvrir une nouvelle fenêtre. La première fenêtre est ouverte avec COM3, tandis que la deuxième avec l'adresse IP et le port du serveur.



Le serveur répond « Hello ». Le message s'affiche dans la fenêtre du client TCP Teraterm. Lorsque vous avez terminé, vous pouvez fermer la fenêtre client TCP.

L'instruction « // client.close(); » est placée en commentaire. Retirer la mise en commentaire, compiler et exécuter à nouveau le programme. Expliquer le résultat obtenu.

Main.ccp() premier serveur TCP :

```
#include "mbed.h"
#include "EthernetInterface.h"
EthernetInterface eth;
char ip[] = "192.168.1.10";
char masque[] = "255.255.255.0";
char passerelle[] = "192.168.1.254";
TCPSocketServer server;
TCPSocketConnection client;
int serveur_port = 2000;
DigitalOut myled(LED1);
Serial pc(USBTX, USBRX);
int main() {
    pc.printf("Hello, I'm happy my Serial port is ok.\n\r");
    if (eth.init(ip,masque,passerelle)!=0) {
        pc.printf("ERREUR INIT ETHERNET\r\n");
        return -1;
    }
    if (eth.connect()!=0) {
        pc.printf("ERREUR CONNECT\r\n");
        return -1;
    }
    pc.printf("L adresse Ip est %s\r\n", eth.getIPAddress());
    pc.printf("IP est OK, vous pouvez continuer\r\n");
    if (server.bind (serveur_port)!=0) {
        pc.printf("Erreur Tcp bind\r\n");
        return -1;
    }
    pc.printf("Succes Tcp Bind\r\n");
    if (server.listen()!=0){
        pc.printf("ERREUR Tcp Listen\r\n");
        return -1;
    }
    pc.printf("Succes Tcp listen\r\n");
    pc.printf("Connectez un client Tcp port 2000...\r\n");
    if (server.accept (client)!=0){
        pc.printf("ERREUR Tcp Listen\r\n");
        return -1;
    }
    pc.printf("Client Tcp connecte\r\n");
    pc.printf("Adresse Ip client %s\r\n",client.get_address());
    client.send("Hello\r\n",strlen("Hello\r\n"));
    // client.close();
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

Nous constatons que tant que le serveur est en attente d'une connexion d'un client, la LED ne clignote pas. Cela est dû au fait que la méthode « accept () » est bloquante. Il n'est plus ensuite possible de connecter un client au serveur. Nous allons voir par la suite comment opérer.

Programmation réseau deuxième serveur.

Vous avez vu précédemment :

- comment créer un serveur,
- accepter une demande connexion d'un client,
- envoyer un message à ce client,
- fermer la connexion avec le client.

Maintenant, il va falloir répéter ceci et permettre de gérer des connexions de clients. Il ne sera cependant possible de connecter qu'un seul client à la fois.

Nous allons dans un premier temps sauvegarder le projet « TP4_prog-reseau-premier-serveur » sous le nom « TP4_prog-reseau-deuxieme-serveur ». Nous utiliserons ce projet dans cette partie.

Résultat demandé : les données en provenance du port série virtuel seront copiées vers l'interface Ethernet en TCP et vice versa.

Certaines fonctions sockets sont bloquantes, pour les rendre non bloquantes, nous allons utiliser la méthode : **client.set_blocking(false, 100); // Timeout after (100ms)**. Attention à la valeur du Timeout, trop long, il donne un temps de latence, trop court il peut y avoir des erreurs de temps de réponse.

Le programme page suivante vous donne une base. Attention la fonction receive() ne renvoie de données que lorsqu'elle reçoit le code de la touche entrée.

Une variable **"bool client_connected = false;"** est créée et mise à false au départ. Elle permet de vérifier si un client est connecté.

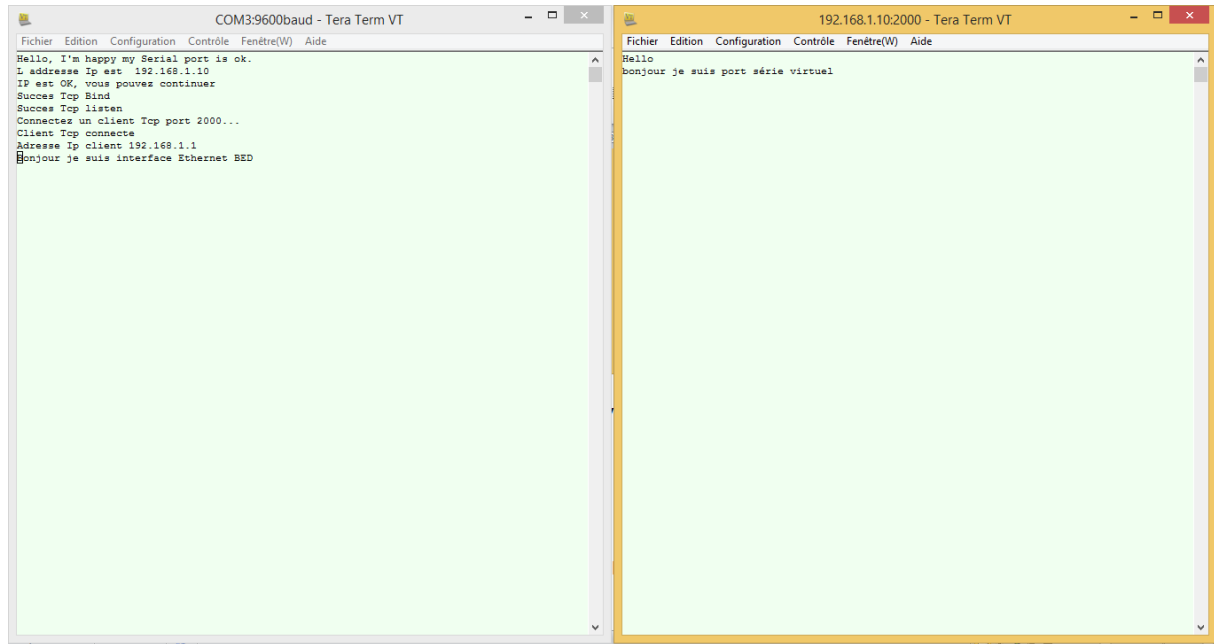
Quelques explications :

Vous regardez la boucle while(1) et établissez un organigramme du programme. Validez le fonctionnement de ce programme. Vous ajoutez au programme l'allumage de la LED1 lors de la connexion avec un client.

Main.ccp() deuxième serveur Tcp :

```
#include "mbed.h"
#include "EthernetInterface.h"
EthernetInterface eth;
char ip[] = "192.168.1.10";
char masque[] = "255.255.255.0";
char passerelle[] = "192.168.1.255";
TCPSocketServer server;
TCPSocketConnection client;
int serveur_port = 2000;
DigitalOut myled(LED1);
Serial pc(USBTX, USBRX);
int main() {
    bool client_connected = false;
    char buffer [100];
    pc.printf("Hello, I'm happy my Serial port is ok.\n\r");
    if (eth.init(ip,masque,passerelle)!=0) {
        pc.printf("ERREUR INIT ETHERNET\r\n");
        return -1;
    }
    if (eth.connect()!=0) {
        pc.printf("ERREUR CONNECT\r\n");
        return -1;
    }
    pc.printf("L adresse Ip est %s\r\n", eth.getIPAddress());
    pc.printf("IP est OK, vous pouvez continuer\r\n");
    if (server.bind (serveur_port)!=0) {
        pc.printf("Erreur Tcp bind\r\n");
        return -1;
    }
    pc.printf("Succes Tcp Bind\r\n");
    if (server.listen()!=0){
        pc.printf("ERREUR Tcp Listen\r\n");
        return -1;
    }
    pc.printf("Succes Tcp listen\r\n");
    pc.printf("Connectez un client Tcp port 2000...\r\n");
```

```
client.set_blocking(false, 100); // Timeout after (100ms)
while(1) {
    if (client_connected==false)
    {
        if (server.accept (client)!=0){
            pc.printf("ERREUR connexion client\r\n"); // erreur
            return -1;
        }
        client_connected = true;
        pc.printf("Client Tcp connecte\r\n");
        pc.printf("Adresse Ip client %s\r\n",client.get_address());
        client.send("Hello\r\n",strlen("Hello\r\n"));
    }
    else
    {
        if ( client.is_connected() )
        {
            if (pc.readable()) // interface virtuel serie recoit donnees
            {
                buffer[0] = pc.getc();
                client.send(buffer,1);
            }
            int nbcaractere = client.receive(buffer,sizeof(buffer));
            if (nbcaractere>0) // interface Ethernet recoit donnees
            {
                buffer[nbcaractere-1] = 0;
                pc.printf("%s",buffer);
            }
        }
        else
        {
            client_connected = false;
            client.close();
            pc.printf("Client deconnecte\r\n");
            pc.printf("Connectez un client Tcp port 2000...\r\n");
        }
    }
}
}
```



Nous voyons le dialogue s'effectuer entre le port virtuel série (écran gauche) et le port Ethernet en TCP (écran droit).

- Nous nous plaçons sur l'écran à gauche (port série virtuel). Nous écrivons et nous constatons que les données s'affichent immédiatement sur l'écran à droite.
- Par contre lorsque nous nous plaçons sur l'écran à droite, les données ne sont pas affichées immédiatement sur l'écran à gauche. Il faut valider avec la touche « enter ».

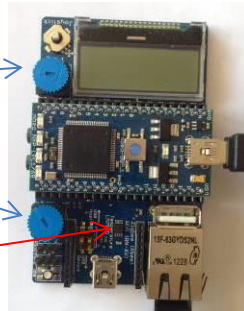
Programmation réseau troisième serveur : station météo communicante.

Nous proposons de réaliser une station météo connectée simplifiée à partir de la carte MBED et de la carte « Mbed Application Board ». Celle-ci possède 2 potentiomètres analogiques Iskra PNZ10ZA (Pot1 à gauche : p19 et Pot2 à droite : p20), qui simuleront pour p19 un capteur de vitesse de vent et pour p20 un capteur d'humidité et un capteur de température I2C LM75B (SCL : p27, SDA : p28 et adresse : 0x90).

Capteur vitesse vent p19

Capteur d'humidité p20

Capteur température LM75



- Vitesse du vent : 0 à 200km/h.

- Humidité : 0 à 100%.

La mise en forme de ces capteurs est réalisée de façon à ce que la variation d'un capteur sur sa plage d'acquisition corresponde à une variation du CAN sur sa plage de conversion.

La station météo étant communicante, nous pourrions l'interroger à distance en Ethernet. Les commandes seront les suivantes :

Commande	Action et Grandeur	Message renvoyé : format
't' suivi de CR (retour ligne)	Lire température	«t=xx.xC» suivi de 'cr' 'lf'
'v' suivi de CR (retour ligne)	Lire vitesse vent	«v=xxx.xkm/h» suivi de 'cr' 'lf'
'h' suivi de CR (retour ligne)	Lire humidité	«h=xx.x%» suivi de 'cr' 'lf'
'r' suivi de CR (retour ligne)	Reset logiciel de la carte MBED	
'?' suivi de CR (retour ligne)	Renvoie une aide sur les commandes.	

Nous pouvons imaginer qu'une ligne de commande, qui comprend plusieurs codes de commande s'exécute comme si plusieurs commandes avaient été envoyées séparément.

Phase 1 : intégration des mesures et validation par le port virtuel série.

Nous allons dans un premier temps sauvegarder le projet «TP4_prog-reseau-deuxieme-serveur» sous le nom TP4_prog-reseau-troisieme-serveur-phase1». Nous utiliserons ce projet dans cette partie.

Dans cette partie nous allons mettre en fonctionnement le capteur et les entrées analogiques. Les résultats seront affichés sur le port série virtuel.

Nous allons :

- Instancier les objets correspondant aux potentiomètres.
- Ajouter la librairie du LM75.
- Instancier un objet LM75.

Dans la boucle while :

- Supprimer les lignes, qui correspondent au clignotement des LEDS.
- Acquérir les tensions produites par chaque potentiomètre et la température produite par le LM75.
- Calculer les tensions et la température.
- Transmettre sur le port virtuel série ces grandeurs sous forme d'une chaîne de caractère du type : t=xx.xC,v=xxx.xkm/h,h=xx.x'CR''LF'

Importation de la librairie du LM75.

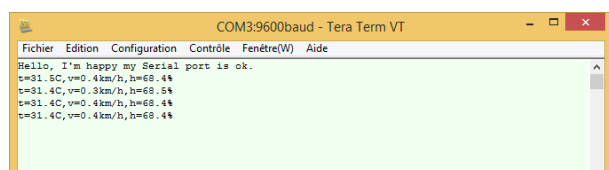
Name	Tags	Author	Imports	Modified	Description
☆ LM75B	I2C LM75B	Chris Styles	1585	26 Oct 2012	A simply library for the LM75B I2C temperature sensor
☆ LM75B	Celsius driver I2C LM75B nxp s	Neil Thiessen	475	30 May 2014	A feature complete driver for the LM75B temperature se
☆ LM75A	I2C LM75A	Edoardo De Marci	46	15 Aug 2014	Library for the LM75A Temperature Sensor
☆ LM75	LM75	Matthias Leon	1	20 Mar 2015	Library for using LM75.
★ PCT2075	LM75B PCT2075 sensor Tempe	Tedd OKANO	2	05 Mar 2015	Component class library for PCT2075 and LM75B. The PC
☆ Lmic_1v2	network protocol wireless	www.reberkz	1	27 Mar 2015	IBM Lmic_1v2 for M07E_L152RC
☆ LM77	I2C LM77 sensor Temperature	Wim Huiskamp	0	10 Jan 2015	LM77 Temperature sensor with I2C interface. Provides t

Vous importez la librairie LM75B PCT2075. Elle est commune à ces 2 composants. Nous mettons en commentaire les instructions placées dans la fonction main().

Le fichier main.ccp devient :

```
include "mbed.h"
#include "EthernetInterface.h"
#include "LM75B.h"
LM75B temp_sensor(p28, p27);
AnalogIn Pot1(p19);
AnalogIn Pot2(p20);
EthernetInterface eth;
char ip[] = "192.168.1.10";
char masque[] = "255.255.255.0";
char passerelle[] = "192.168.1.255";
TCPSocketServer server;
TCPSocketConnection client;
int serveur_port = 2000;
DigitalOut myled(LED1);
Serial pc(USBTX, USBRX);
int main() {
    pc.printf("Hello, I'm happy my Serial port is ok.\n\r");
    while(1) {
        pc.printf( "t=%2.1fC,v=%3.1fkm/h,h=%2.1f%%\r\n", (float)temp_sensor.read(),Pot1.read()*200,Pot2.read()*100);
        wait( 1 );
    }
    /* bool client_connected = false; à partir d'ici tout est placé en commentaire jusqu'à la fin de la fonction main.
    char buffer [100];
    */
}
```

Le résultat est conforme à ce qui est attendu :



Nous pouvons passer à la phase 2.

Phase 2 : mise en place du serveur Ethernet.

Nous conservons les paramètres Ethernet précédent. Nous allons réaliser un serveur simplifié. Il renverra les mesures sous la même forme que dans la partie Phase 1.

Nous allons sauvegarder le projet «TP4_prog-reseau-troisieme-serveur-phase1» sous le nom TP4_prog-reseau-troisieme-serveur-phase2». Nous utiliserons ce projet dans cette partie.

Nous allons enlever les commentaires à la partie Ethernet et en placer à la partie ajoutée précédemment. Le fichier main.ccp devient :

```
#include "mbed.h"
#include "EthernetInterface.h"
#include "LM75B.h"
LM75B temp_sensor( p28, p27 );
AnalogIn Pot1(p19);
AnalogIn Pot2(p20);
EthernetInterface eth;
char ip[] = "192.168.1.10";
char masque[] = "255.255.255.0";
char passerelle[] = "192.168.1.255";
TCPSocketServer server;
TCPSocketConnection client;
int serveur_port = 2000;
DigitalOut myled(LED1);
Serial pc(USBTX, USBRX);
int main() {
    /* pc.printf("Hello, I'm happy my Serial port is ok.\n\r");
    while(1) {
        pc.printf("t=%2.1fC,v=%3.1fkm/h,h=%2.1f%%\r\n",
            (float)temp_sensor.read(),Pot1.read()*200,Pot2.read()*100);
        wait( 1 );
    }*/
    bool client_connected = false;
    char buffer [100];
    char commande[100];
    char *cmde;
    pc.printf("Hello, I'm happy my Serial port is ok.\n\r");
    if (eth.init(ip,masque,passerelle)!=0) {
        pc.printf("ERREUR INIT ETHERNET\r\n");
        return -1;
    }
    if (eth.connect()!=0) {
        pc.printf("ERREUR CONNECT\r\n");
        return -1;
    }
    pc.printf("L adresse Ip est %s\r\n", eth.getIPAddress());
    pc.printf("IP est OK, vous pouvez continuer\r\n");
    if (server.bind (serveur_port)!=0) {
        pc.printf("Erreur Tcp bind\r\n");
        return -1;
    }
    pc.printf("Succes Tcp Bind\r\n");
    if (server.listen()!=0){
        pc.printf("ERREUR Tcp Listen\r\n");
        return -1;
    }
    pc.printf("Succes Tcp listen\r\n");
    pc.printf("Connectez un client Tcp port 2000...\r\n");
    client.set_blocking(false, 100); // Timeout after (100ms)
    while(1) {
        if (client_connected==false)
        {
            if (server.accept (client)!=0){
                pc.printf("ERREUR connexion client\r\n"); // erreur
                return -1;
            }
            client_connected = true;
            pc.printf("Client Tcp connecte\r\n");
            pc.printf("Adresse Ip client %s\r\n",client.get_address());
            client.send("Hello\r\n",strlen("Hello\r\n"));
        }
        else
        {
            if ( client.is_connected() )
            {
                /* Partie a completer */
            }
            else
            {
                client_connected = false;
                client.close();
                pc.printf("Client deconnecte\r\n");
                pc.printf("Connectez un client Tcp port 2000...\r\n");
            }
        }
    }
}
```

Page précédente, nous voyons à gauche la partie initialiation et à droite la partie exécution du serveur. Il va falloir écrire le programme, ou il est indiqué **/* Partie a completer */**.

Dans cette partie nous avons à :

- Tester si le client a envoyé des données.
- Envoyer au client les mesures suivant le même format qu'en phase 1.

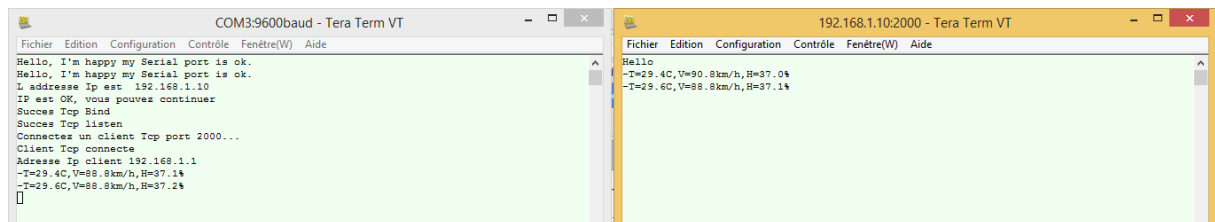
Remarque : l'objet client ne possède pas de méthodes de type printf, qui permettent de formater des données. Il va falloir passer par une phase intermédiaire.

- Formatage avec la fonction sprintf dans un tableau de caractère.
- Envoi vers le client avec la méthode send.

```
int nbcaractere = client.receive(commande,sizeof(commande));
if (nbcaractere>0) // interface Ethernet recoit donnees
{
    commande[nbcaractere-1] = 0;
    pc.printf("-T=%3.1fC,V=%3.1fkm/h,H=%3.1f%%\r\n", (float)temp_sensor.read(),Pot1.read()*200,Pot2.read()*100);
    sprintf(buffer, "-T=%3.1fC,V=%3.1fkm/h,H=%3.1f%%\r\n", (float)temp_sensor.read(),Pot1.read()*200,Pot2.read()*100);
    client.send(buffer,strlen(buffer));
}
```

Attention : le client envoie les données lorsque la fenêtre à droite est active et à chaque appui sur la touche « enter »

Les tests nous montrent un ensemble fonctionnel.



Il ne reste plus qu'à intégrer la reconnaissance des commandes.

Phase 3 : interpréteur de commande.

Nous allons sauvegarder le projet «TP4_prog-reseau-troisieme-serveur-phase2» sous le nom TP4_prog-reseau-troisieme-serveur-phase3». Nous utiliserons ce projet dans cette partie.

Nous vous proposons le programme page suivante à intégrer dans la partie à compléter.

```
int nbcaractere = client.receive(commande,sizeof(commande));
if (nbcaractere>0) // interface Ethernet recoit donnees
{
    commande[nbcaractere-1] = 0;
    cmde = commande;
    pc.printf( "NB caract recus %d Mesure T=%3.1fC,V=%3.1fkm/h,H=%3.1f%%\r\n",nbcaractere,
(float)temp_sensor.read(),Pot1.read()*200,Pot2.read()*100);
    while ((*cmde)!=0)
    {
        switch (*cmde) {
            case 't' :
                sprintf(buffer,"T=%2.1fC\r\n", (float)temp_sensor.read());
                client.send(buffer,strlen(buffer));
                break;
            case 'v' :
                sprintf(buffer,"V=%3.1fkm/h\r\n", Pot1.read()*200);
                client.send(buffer,strlen(buffer));
                break;
            case 'h' :
                sprintf(buffer,"H=%3.1f%%\r\n", Pot2.read()*100);
                client.send(buffer,strlen(buffer));
                break;
            case 'r' :
                NVIC_SystemReset();
                break;
            case 0 : break;
            case '\r' : break;
            case '\n' : break;
            default:
                sprintf(buffer,"Commande inconnue %c - T=%2.1fC,V=%3.1fkm/h,H=%3.1f%%\r\n",*cmde,
(float)temp_sensor.read(),Pot1.read()*200,Pot2.read()*100);
                client.send(buffer,strlen(buffer));
        }
        cmde++;
    }
}
```

Nous vous proposons un VI « client Labview » uniquement pour la mesure de la température. Vous trouvez ce client sur le site du BTS SN à l'adresse :

<http://lyceehugobesancon.org/btssn/spip.php?article1177>

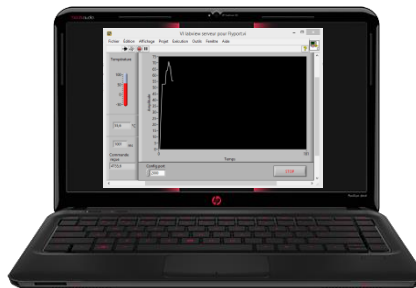
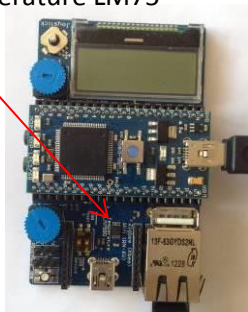
Vous testez ce programme après avoir lancé le programme MBED.

Attention pour le cas ou vous utilisez 2 Pc vous utilisez le VI Labview sur le Pc de développement.

Programmation reseau client : capteur temperature.

Nous proposons de réaliser capteur de temperature autonome communiquant par le reseau. Les mesures s'effectuent avec une frequence programmable ayant pour base la minute. Dans un premier temps nous allons utiliser la liaison Ethernet de la carte MBED. Elle pourra etre par la suite remplacee par une liaison Wifi.

Capteur temperature LM75



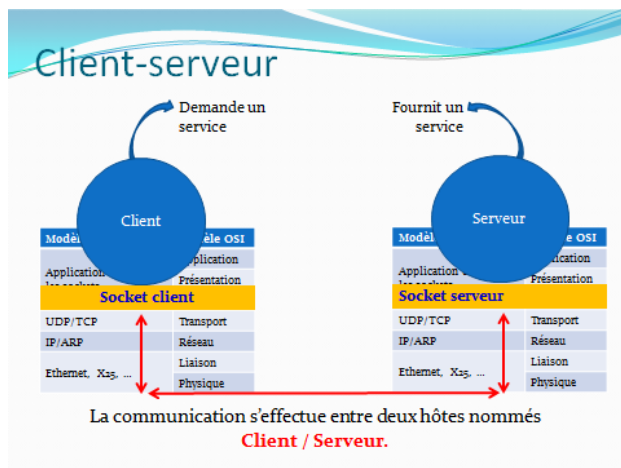
Ordinateur portable avec port Ethernet et Wifi

Teraterm et Labview

Client-serveur :

La carte MBED relève la temperature à l'aide du capteur LM75 et la transmet au PC. Se pose le probleme de qui sera client et qui sera serveur.

- La carte MBED consomme de 60 à 120mA. Si la carte MBED est utilisée en serveur, elle devra etre en permanence à l'écoute. Au contraire si elle est utilisée en client, elle pourra etre en veille entre 2 mesures. La solution carte MBED client et PC serveur permettra une plus grande autonomie.



- Le PC a le rôle de serveur et utilisera pour cela un serveur sur Labview.

- La carte MBED a le rôle de client. Nous allons la programmer pour cela.

Les paramètres Ip sont identiques à ceux définis aux activités précédentes.

Configuration Ip

Adresse reseau	192.168.1.0
Masque sous reseau	255.255.255.0
Passerelle	192.168.1.254
PC	192.168.1.1
Carte MBED	192.168.1.10

Attention l'adressage reseau coté Ethernet doit etre différent de celui coté Wifi.

Configuration Tcp

Port utilisé	2000
--------------	------

Présentation du VI « serveur Labview ».

Le VI « serveur Labview » est simple, vous pouvez le charger sur le site du BTS SN du lycée Victor Hugo à l'adresse : <http://lyceehugobesancon.org/btssn/spip.php?article1177>.

Face avant :

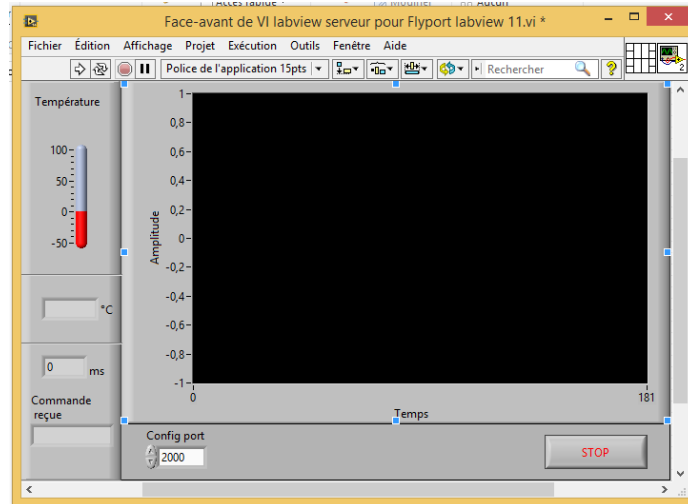
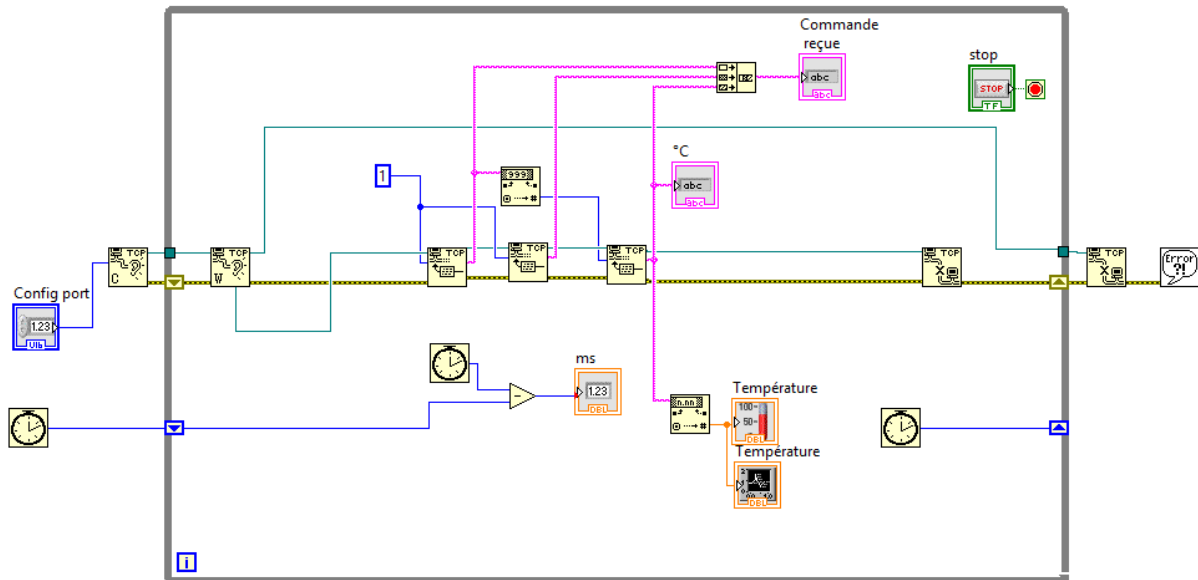


Diagramme :



Le format des données est le suivant :

I	T	x	x	,	x
Chaîne caractère température					
Caractère 'T' indique température					
Longueur de la chaîne de caractère xx,x ici 4					

Par exemple pour une température mesurée de : 16,687°C la chaîne transmise est 4T16.6 (16.6 a une longueur de 4 caractères). Attention le symbole décimal est la virgule.

Phase 1 :

Nous allons dans un premier temps sauvegarder le projet «TP4_prog-reseau» sous le nom «TP4_prog-reseau_client». Nous utiliserons ce projet dans cette partie.

Nous ajoutons la librairie PCT2075 en la copiant depuis le projet « TP4_prog-reseau-troisieme-serveur-bis-phase3 ».

Nous allons ajouter au programme les lignes suivantes :

- TCPSocketConnection client; // instancier un objet client
- #define Ip_serveur "192.168.1.1" // définir les paramètre du serveur
- #define Port_serveur 2000
- #include "LM75B.h" //ajouter la librairie du capteur
- LM75B temp_sensor(p28, p27); //instancier un objet tem_sensor.

Nous allons pouvoir utiliser le socket client. Nous disposons des méthodes :

- int TCPSocketConnection::connect(const char* host, const int port) : permet d'ouvrir la connexion avec le serveur. Si le résultat est négatif l'opération s'est mal déroulée.
- int TCPSocketConnection::send(char* data, int length) : permet d'envoyer un message au serveur.
- ...
- int Socket::close(bool shutdown) : permet de fermer la connexion.

Nous allons utiliser ces différentes méthodes, le programme proposé est le suivant :

<pre>#include "mbed.h" #include "EthernetInterface.h" EthernetInterface eth; char ip[] = "192.168.1.10"; char masque[] = "255.255.255.0"; char passerelle[] = "192.168.1.255"; TCPSocketConnection client; char Ip_serveur[] = "192.168.1.1"; int Port_serveur = 2000; #include "LM75B.h" LM75B temp_sensor(p28, p27); DigitalOut myled(LED1); Serial pc(USBTX, USBRX);</pre>	<pre>int main() { pc.printf("Hello, I'm happy my Serial port is ok.\n\r"); if (eth.init(ip,masque,passerelle)!=0) { pc.printf("ERREUR INIT ETHERNET\r\n"); return -1; } if (eth.connect()!=0) { pc.printf("ERREUR CONNECT\r\n"); return -1; } pc.printf("L adresse Ip est %s\r\n", eth.getIPAddress()); pc.printf("IP est OK\r\n "); while(1) { pc.printf("\r\nPatientez demande connexion au serveur\r\n "); pc.printf("- Ip adresse %s\r\n ",Ip_serveur); pc.printf("- Ip adresse %d\r\n ",Port_serveur); if (client.connect(Ip_serveur,Port_serveur)< 0) { pc.printf("Erreur : Socket non connecte\r\n"); } else { pc.printf("Serveur connecte connecte\r\n"); char buffer[] = "4T16,9"; client.send(buffer,strlen(buffer)); pc.printf("Message envoye\r\n"); client.close(); pc.printf("Connexion serveur ferme\r\n"); } pc.printf("Patientez 1s\r\n"); wait(1); } }</pre>
---	---

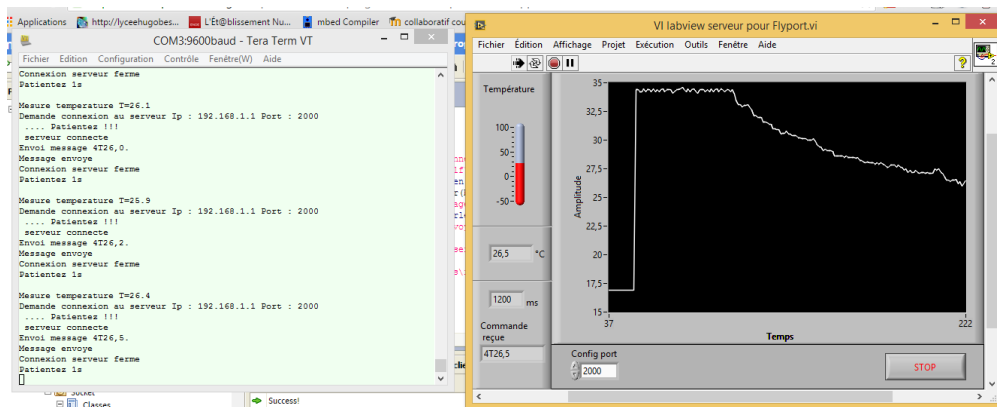
Vous validez le fonctionnement du client. (Ajouter ici une copie d'écran de Teraterm coté USB virtual).

Phase 2 :

Vous modifiez le programme suivant le cahier des charges. Le programme vous est donné ici mais nous vous conseillons de le réaliser vous-même.

```
#include "mbed.h"
#include "EthernetInterface.h"
EthernetInterface eth;
char ip[] = "192.168.1.10";
char masque[] = "255.255.255.0";
char passerelle[] = "192.168.1.255";
TCPSocketConnection client;
char Ip_serveur[] = "192.168.1.1";
int Port_serveur = 2000;
#include "LM75B.h"
LM75B temp_sensor( p28, p27 );
DigitalOut myled(LED1);
Serial pc(USBTX, USBRX);

int main() {
    pc.printf("Hello, I'm happy my Serial port is ok.\n\r");
    if (eth.init(ip,masque,passerelle)!=0) {
        pc.printf("ERREUR INIT ETHERNET\r\n");
        return -1;
    }
    if (eth.connect()!=0) {
        pc.printf("ERREUR CONNECT\r\n");
        return -1;
    }
    pc.printf("L adresse Ip est %s\r\n", eth.getIPAddress());
    pc.printf("IP est OK, vous pouvez continuer\r\n ");
    char buffer [100];
    while(1) {
        pc.printf("\r\nMesure temperature T=%3.1f\r\n",(float)temp_sensor.read());
        pc.printf("Demande connexion au serveur Ip : %s Port : %d\r\n ",Ip_serveur,Port_serveur);
        pc.printf(".... Patientez !!!\r\n ");
        if (client.connect(Ip_serveur,Port_serveur)< 0)
        {
            pc.printf("Erreur connexion au serveur\r\n");
            client.close();
        }
        else
        {
            pc.printf("serveur connecte\r\n");
            sprintf(buffer,"xT%3.1f",(float)temp_sensor.read());
            buffer[0] = char(strlen(buffer)-2) + '0'; // calcul longueur chaine et traduit ASCII
            char *ptr;ptr = strchr(buffer,');*ptr = ','; // remplace point par virgule
            pc.printf("Envoi message %s.\r\n",buffer);
            client.send(buffer,strlen(buffer));
            pc.printf("Message envoye\r\n");
            client.close();
            pc.printf("Connexion serveur ferme\r\n");
        }
        pc.printf("Patientez 1s\r\n");
        wait(1);
    }
}
```

**Remarques :**

- pour la version finale, il faudra penser à enlever les commentaires, qui concernent la liaison virtuelle série.
- Afin de limiter la consommation du client, Il est possible de le mettre en veille le client après chaque échange et de le réveiller avec la RTC.