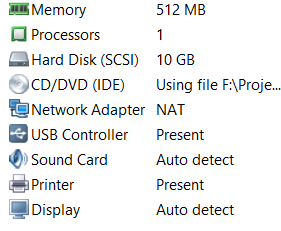
Internet des Objets – NodeJS

Joncour Grégoire – Neves Loïc – Kabasele Quentin – Losio Emy-Julie

Une petite application web qui récupère les informations de notre objet.

1. Installation des machines virtuelles
   1. **Configuration du Serveur**



La machine du serveur est munie de 512MB de mémoire, un processeur et un disque dur (SCSI) de 10 GB. Elle est également connectée en Nat, elle a accès au LAN grâce à notre machine et utilise notre adresse IP. Notre machine fait donc office de serveur DHCP.

Selon les goûts, on a utilisé comme système d’exploitation Debian ou Ubuntu.



* 1. **Installation du SSH**
     1. **Qu’est-ce que le SSH**

Le SSH (**Protocole Secure Shell**) permet de réaliser des opérations à distance. Il permet à un client d'ouvrir une session interactive sur une machine distante (serveur) afin d'envoyer des commandes ou des fichiers de manière sécurisée.

L’inconvénient en utilisant le protocole Telnet pour accéder à la machine distante, est que les informations échangées circulent en clair notamment le login et le mot de passe. Il y a donc la possibilité d’écouter le trafic et d’obtenir les données.

Le protocole SSH répond à cette problématique en permettant d’accéder à une machine à travers une communication chiffrée appelée **tunnel**.

* + 1. **Mise en place sur Debian**

Nous avons tapé les commandes suivantes:

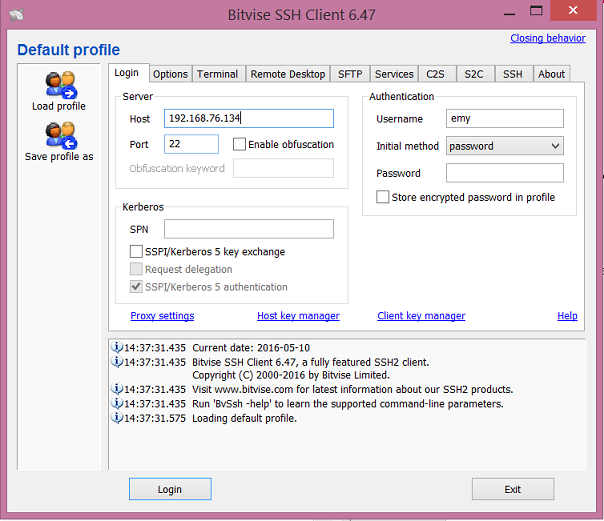
* Su
* apt-get install sudo
* nano/etc/sudoers

Ceci est la modification du fichier sudoers:

* debian ALL=(ALL) ALL -> CntrlX -> exit
* sudo ifconfig
* root

Une fois ceci effectué nous avons installé Bitvise SSH Client, un SSH et SFTP client pour Windows. Cela ressemble à ceci:

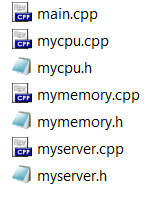
* add ip -> eth0
* debian
* password
* root
* login
* accept and save
* sudo apt-get update



Nous mettons une adresse Host, un Username et un Password.

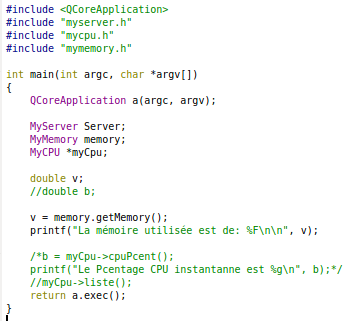
1. Programmation de l’application
   1. **Taux d’occupation CPU et de la mémoire**

Pour calculer le taux d'occupation du CPU et de la mémoire l'architecture du projet se fait comme ceci :



Elle est constituée de 4 classes, Main, Cpu, Memory et Server. Décortiquons-les plus attentivement :

**Main:** possède les 3 instances des 3 autres classes (Cpu, Memory et Server). Il teste les méthodes afin d'afficher les taux d'occupation du CPU et de la mémoire.



**MyCpu**:

* .h:
  + Déclaration de 2 méthodes "cpuPcent" pour le taux d'occupation du CPU et "liste" pour afficher la liste des processus actifs en détail.
* .cpp:
  + Instanciation des 2 méthodes + constructeur

**MyMemory**:

* .h:
  + Déclaration d'une méthode pour récupérer le taux d'occupation de la mémoire.
* .cpp:
  + Instanciation de cette méthode

**MyServer**:

* .h:
  + Déclaration d'une instance de "QTcpServer" afin de créer un serveur par la suite.
  + Une méthode "newConnection" qui servira à créer la connexion Client/Server.
* .cpp:
  + Instanciation de cette méthode ainsi qu'une modification du constructeur par défaut

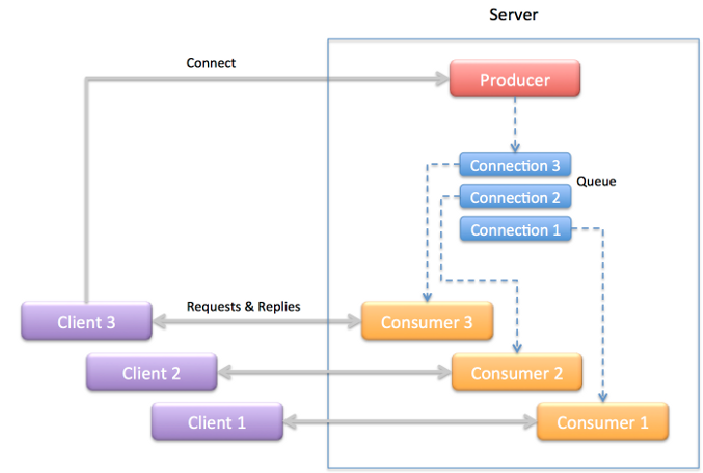
1. Méthode et Protocole utilisés
   1. **Accès concurrentiels**

Il existe deux méthodes pour gérer les accès concurrentiels sur cet objet.

* La méthode UDP qui utilise un serveur UDP qui possède une adresse IP. Ce serveur envoie une requête via la passerelle qui est destinée au client. Lorsque le client veut interroger le serveur, la passerelle ne sait pas où renvoyer cette nouvelle requête qui est destinée au serveur UDP.

Cette méthode n'est donc pas appropriée car la requête ne parvient pas au destinateur.

* La méthode TCP qui met en relation un client est un serveur qui fait fonctionner les capteurs. Le client peut interroger le serveur sur les différents capteurs situés dans les différents processus.
  + Multicast : le client interroge un à un les capteurs et a la possibilité de lecture.
  + Multithreading : lance simultanément plusieurs processus de traitement.



Plusieurs options s'offrent donc à nous :

1) Un sémaphore est une structure de données qui est maintenue par le système d’exploitation et contient :

* Un entier qui stocke la valeur, positive ou nulle, du sémaphore.
* Une queue qui contient les pointeurs vers les threads qui sont bloqués en attente sur ce sémaphore.

Message Queue

On a décidé d'utiliser les message Queue que l'on a trouvé plus adapté à notre cas. Cela évite les conflits puisque les messages ont une clef. Le fait qu'ils soient rangés en fil d'attente évite tous conflits.

* 1. **Accès à l'interface (Q4)**

Le serveur étant le soubassement de notre projet, nous avons décidé de mettre en place un protocole d’interrogation afin de l'exploiter au mieux notre.

Pourquoi les accès concurrentiels ?

La création d’un protocole serveur est nécessaire pour notre objet.

En effet, il est impératif de prévoir un certain nombre de fonctionnalités pour la connexion d’un client sur le serveur.

À la connexion, le client aura la possibilité de récupérer des informations (CPU, mémoire, flux entrants et sortants sur les connecteur virtuels Ethernet) :

• Lire les taux d’occupation

• Modifier les taux d’occupation

La modification de l’objet ne peut pas être fait en même temps par deux clients.

Pour gérer cela on utilise un Mutex (exclusion mutuelle).

Sachant que nous avons plusieurs manières de mettre en place notre système d’exclusion mutuelle :

• Le sémaphore

• message queue (la file d’attente de message) IPC

1. L’IHM en QT/C++



* 1. **Menu principal "Tweak info"**

Comme nous l'avons vu précédemment, notre protocole est simple lors de sa connexion au serveur. Le client a deux possibilités qui s'offre à lui. Ces deux possibilités sont: la récupération des données tweak et la modification d'une donnée. Pour ce faire nous avons donc crée une nouvelle application utilisant le QT widget. Puis j'ai ajouté "QT += network" dans le fichier IOT\_Projet.pro.

Cela fournit un ensemble d'API pour les applications de programmation qui utilisent le protocole TCP / IP. Nous avons donc procédé à la création de mes deux boutons, ainsi que l'ajout de la fonction "clicked()". Puis l'instanciation de la classe serveur qui contient les informations.

* 1. **Flux Ethernet**

Nous commençons sur Debian.

Pour commencer nous avons cherché dans le man 5 proc ("/ Proc" - Process information pseudo-filesystem) les éléments nécessaires à la création de notre flux Ethernet. Les outils dont nous avions besoin sont dans "/proc/net/dev".

Par la suite on utilise la commande cat /proc/net/dev pour afficher le fichier.

Voici ce que l'on obtient:

Inter-| Receive | Transmit

face |bytes packets errs drop fifo frame compressed multicast|bytes packets errs drop fifo colls carrier compressed

lo: 2776770 11307 0 0 0 0 0 0 2776770 11307 0 0 0 0 0 0

eth0: 1215645 2751 0 0 0 0 0 0 1782404 4324 0 0 0 427 0 0

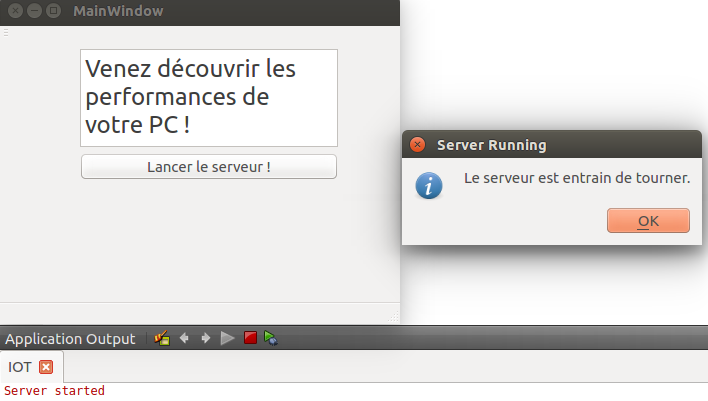
ppp0: 1622270 5552 1 0 0 0 0 0 354130 5669 0 0 0 0 0 0

tap0: 7714 81 0 0 0 0 0 0 7714 81 0 0 0 0 0 0

Dans un premier temps j'ai affiché le fichier au complet. Seule la partie Ethernet nous intéresse donc on affiche la ligne eth0. Pour ce faire, il y a la mise en place de plusieurs variables afin d'afficher les catégories, l'interface et la valeur. Ensuite, nous cherchons à isoler les valeurs des Bytes reçu et transmis afin de pouvoir faire notre flux Ethernet. En conclusion, nous prenons nos deux mesures à deux intervalles différents et faisons une soustraction des valeurs temps 1 ainsi que les valeurs temps 2.

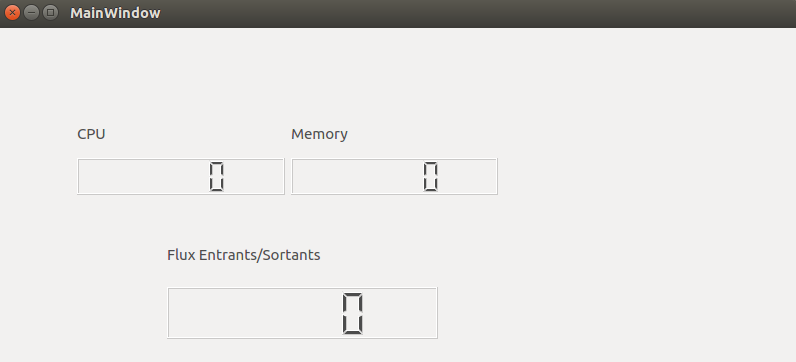
* 1. **Interface de commande**

Nous commençons par le lancement du serveur à partir d'une interface graphique (cf: Code).



Nous voulions que l'interface ait plusieurs fenêtres fonctionnelles telles que:

* Les performances : fenêtre dans laquelle les taux (CPU/Memoire/Flux) devait apparaître dans un premier temps



* Les choix: fenêtre dans laquelle l'utilisateur choisit de récup ses Taux de charge ou bien de modifier les mesures



1. Mise en place de l’application Web
   1. **NodeJS et les sockets**

****

* + 1. **Qu’est-ce que les sockets ?**

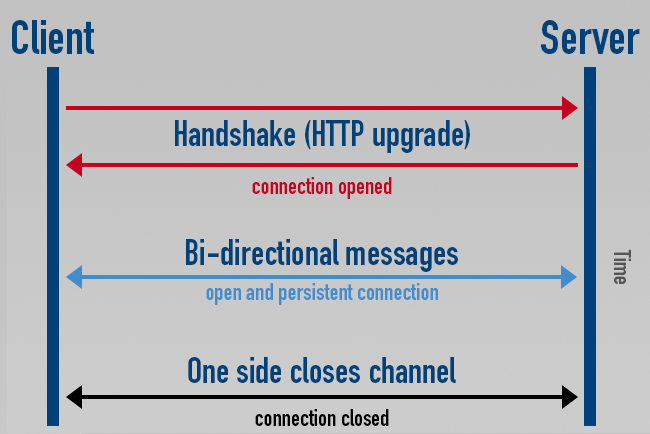
Les WebSockets permettent de créer des applications temps-réel sur le web.

Dans le Web classique, un client envoie une requête à un serveur qui lui répond en lui envoyant les données demandées ou bien une erreur. Afin de voir si de nouvelles données sont arrivées il faudrait donc recharger la page.

:

Le protocole WebSocket autorise lui la communication bidirectionnelle entre le client et le serveur. En effet le serveur peut envoyer directement les données au client sans que celui-ci est à effectuer une requête (et vice-versa).

* + 1. **Comment ça fonctionne ?**



C’est très simple, le client envoie un "Handshake" au serveur pour le prévenir de l'ouverture d'une connexion WebSocket avec. Ce "Handshake" est en fait une **requête** **HTTP**. Si le serveur accepte la connexion entre eux est établie et celle-ci est basée sur le protocole **TCP**. Elle autorise la transmission de message bidirectionnels et reste ouverte jusque l'un des deux interlocuteurs décide de la clore.

* + 1. **Socket.IO**

BLALALALALALA

* 1. **Explication du code**