*Années 2015 - 2016*

Membres du groupe :

Over Yaren

Bernard Florian

Golitin Alexandre

Michaud Théophile

Projet Transversal

Cours Internet des Objets-NodeJS

Sommaire

1. Partie Internet des Objets
2. Introduction
3. Simulation de système embarqué
4. Récupération des mesures
5. Gestion des accès concurrentiels sur l’objet
6. Accès à l’interface du système embarqué
7. Accès à l’objet à distance
8. Partie NodeJS
9. Introduction
10. Interface graphique
11. Les problèmes rencontrés
12. Conclusion
13. Partie Internet des Objets
14. Introduction

Notre projet consiste à réaliser une application comme celles qui sont nécessaires pour gérer et accéder à des objets sur le réseau Internet.

Pour cela, nous avons différentes étapes à produire que nous allons parler dans les parties suivantes.

1. Simulation de système embarqué

Premièrement nous devons réaliser une simulation de système embarqué avec une machine virtuelle sous Linux, sans interface graphique pour être dans la même situation qu'un système embarqué.

Pour cela nous avons créé une machine virtuelle Debian sous Linux en mode console.

Voici la configuration que nous allons effectuer :

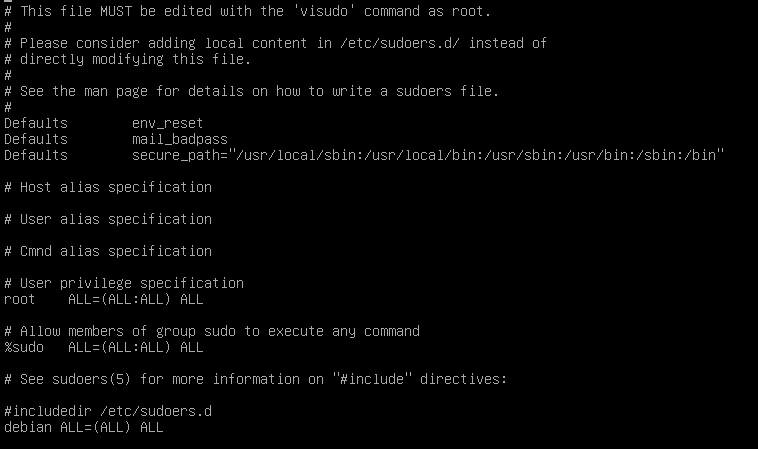
**$ su**

**$ apt-get install sudo**

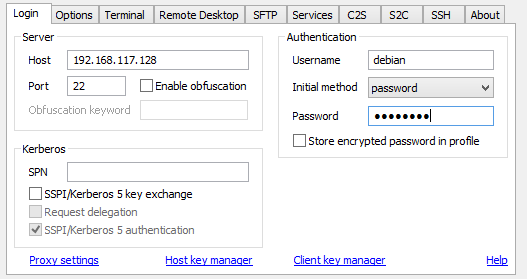
**$ sudo visudo** *-> pour modifier le fichier /etc/sudoers*

**$ debian ALL=(ALL) ALL**

Le module de stratégie sudoers détermine les privilèges sudo d'un utilisateur. Il est le plugin politique de sudo par défaut. La politique est entraînée par le fichier /etc /sudoers que nous allons modifier.



Nous allons ajouter SSH pour s’y connecter à distance. Nous allons utiliser Bitvise SSH Client qui va permettre d’établir des connexions à des serveurs SSH.

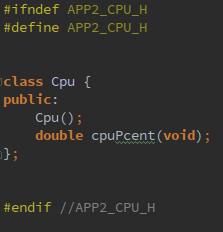
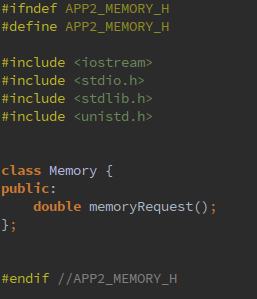


Une fois les paramètres entrée nous allons nous loger.

1. Récupération des mesures

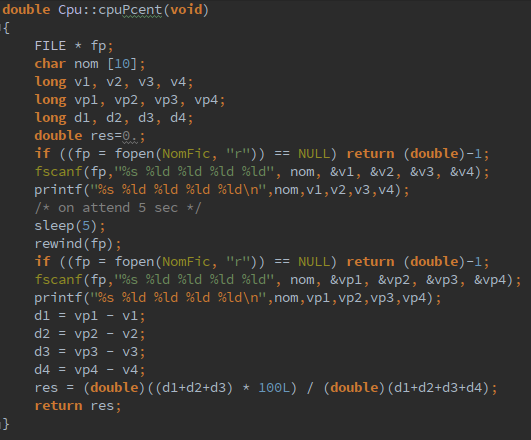
Deuxièmement, nous devions programmer une application qui effectue à intervalle irrégulier des mesures tels que le taux d’occupation du CPU de va machine virtuelle et le taux d’occupation de la mémoire.

Pour cela, nous avons créé une deuxième machine Ubuntu dans laquelle nous avons installé l’IDE Clion pour pouvoir programmer.



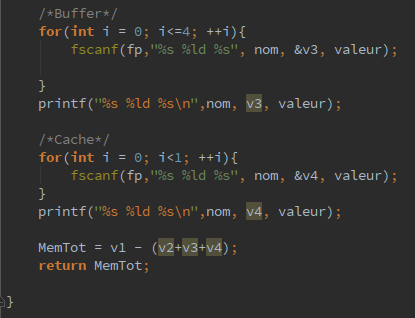
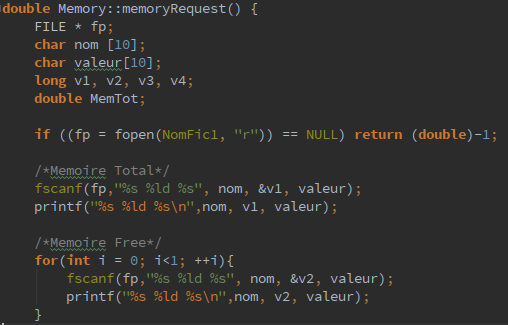
Class Memory

Class Cpu



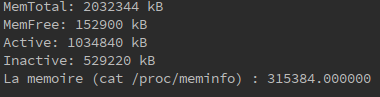
Fonction qui accède à l’interface et calcule le pourcentage CPU

Récupère avec 5s d’intervalle



Fonction qui accède à l’interface et calcule le pourcentage de mémoire

Calcule de la moyenne

C:\Users\yaren\Desktop\Capture.PNGRésultats :

1. Gestion des accès concurrentiels sur l’objet

Pour gérer les accès concurrentiels sur cet objet via le réseau TCP/IP, nous avons défini une méthode qui est d’utiliser un processeur unique au lieu d’en utiliser plusieurs ce qui évitera de créer des conflits. Il sera le seul à gérer les modifications des fichiers.

Sinon dans le cas de l’utilisation de plusieurs processeurs, nous utiliserons les messages queue avec l’ensemble des outils IPC pour gérer les messages.

Les mécanismes d’IPC permettent de faire communiquer ou de synchroniser n’importe quel couple de processus de la même machine. Parmi ces trois mécanismes on trouve les files de messages, les segments de mémoire partagée et les sémaphores.

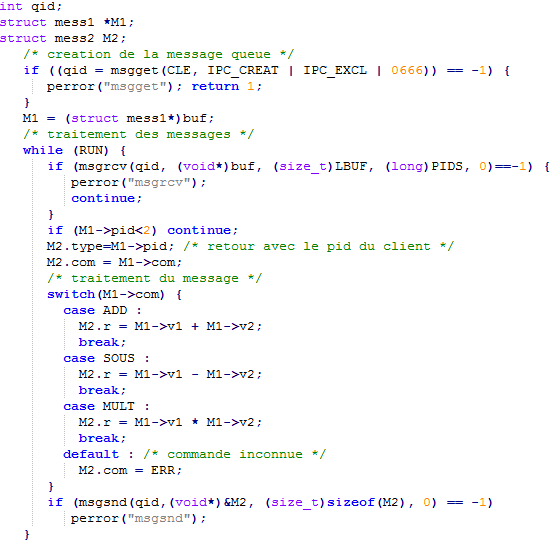
Les messages queue permettent au processus d’envoyer et de recevoir des messages et des messages de file d'attente pour le traitement dans un ordre arbitraire.

Un processus serveur peut diriger le trafic des messages entre les clients sur sa file d’attente en utilisant le processus client PID que le type de message car les messages peuvent être affectés d’un type spécifique.

L’IPC a une référence qui est la clé (Key) qui permet à plusieurs processus d’accéder au même objet IPC mais ceci peut créer des problèmes car la clé peut être déjà utilisée par un autre processus ce qui va créer des conflits. Pour cela nous utiliserons une fonction de création automatique de clé unique appelé ftock(). Mais ceci peut aussi créer des problèmes à cause de la référence donnée en paramètre dans la fonction qui doit exister. Il faudra donc utiliser un fichier verrouillé comme référence.

Avant qu’un processus puisse envoyer ou recevoir un message, la file d’attente doit être initialisé via **msgget().** Les opérations pour envoyer et recevoir des messages sont effectués par le **msgsnd ()** et **msgrcv ()** fonctions, respectivement.

Exemple de code :



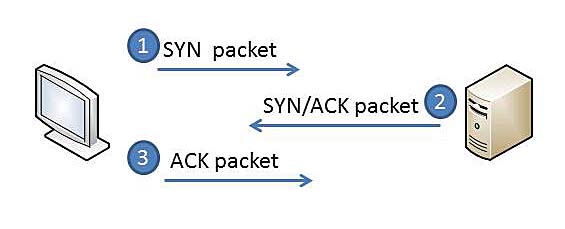
1. Accès à l’interface du système embarqué

Pour accéder à l’interface offerte par le système embarqué, nous avons défini le protocole TCP.

Le protocole TCP fournit un service de transfert de données de haute fiabilité. Il a les fonctionnalités suivantes :

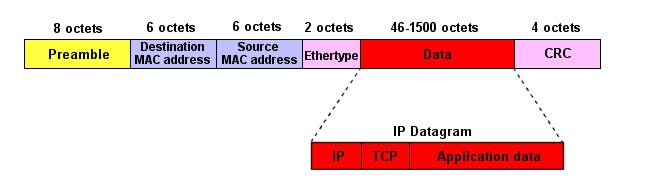
* Transfert de données de base
* Correction d'erreur
* Contrôle de flux
* Multiplexage
* Gestion de connexions
* Priorité et Sécurité

Avant de réaliser des transferts, les processus communiquent d’abord et ouvrent une session :



Lors des transferts des données, chaque paquet contient un numéro de séquence qui va définir l’ordre des données. Cela va permettre de détecter des erreurs car si une donnée manque le client va redemander cette donnée au serveur grâce au numéro de séquence de la donnée.

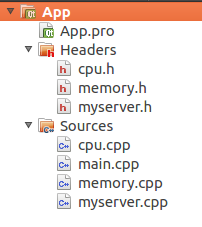
Voici un exemple de trame :



|  |
| --- |
| **Application**  **Telnet** |
| **Transport**  **TCP** |
| **Network**  **IP** |
| **Data Link**  **Ethernet** |
| **Physical** |

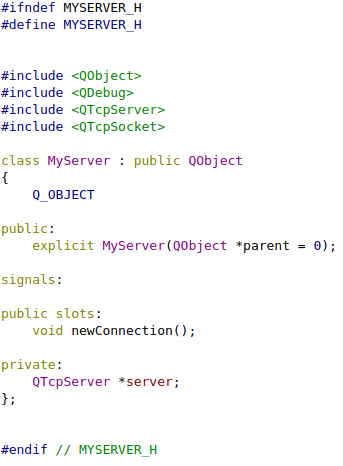
1. Accès à l’objet à distance

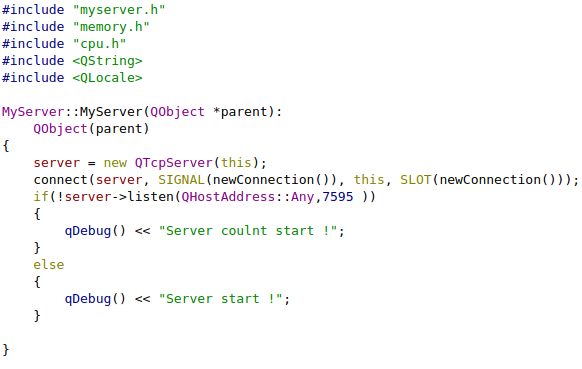
Pour réaliser une IHM en Qt/C++ qui permettra à l’utilisateur d’accéder à l’objet. Pour cela nous avons d’abord installé Qt Creator réalisé l’arborescence ci-dessous :



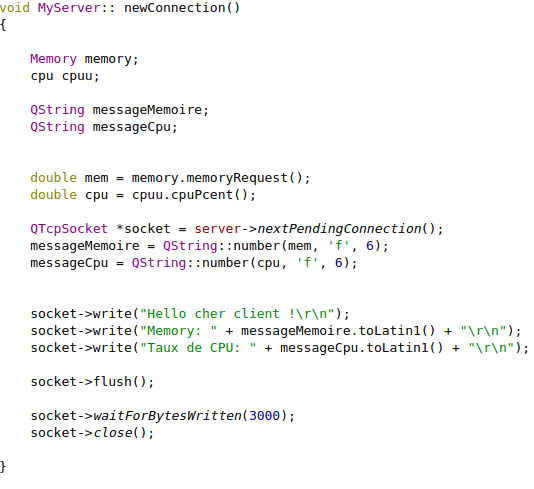
Nous avons ajouté une classe **myserver** au code précédent :

myserver.h



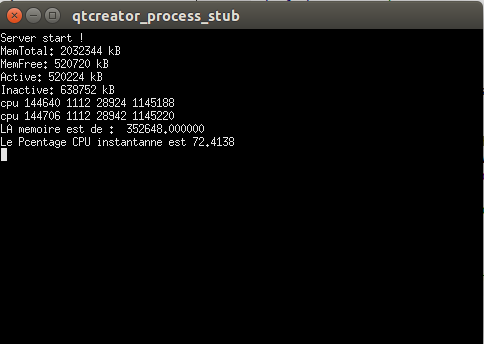


myserver.cpp

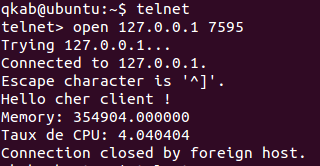


Récupération des mesures

Conversion du type

Résultats :

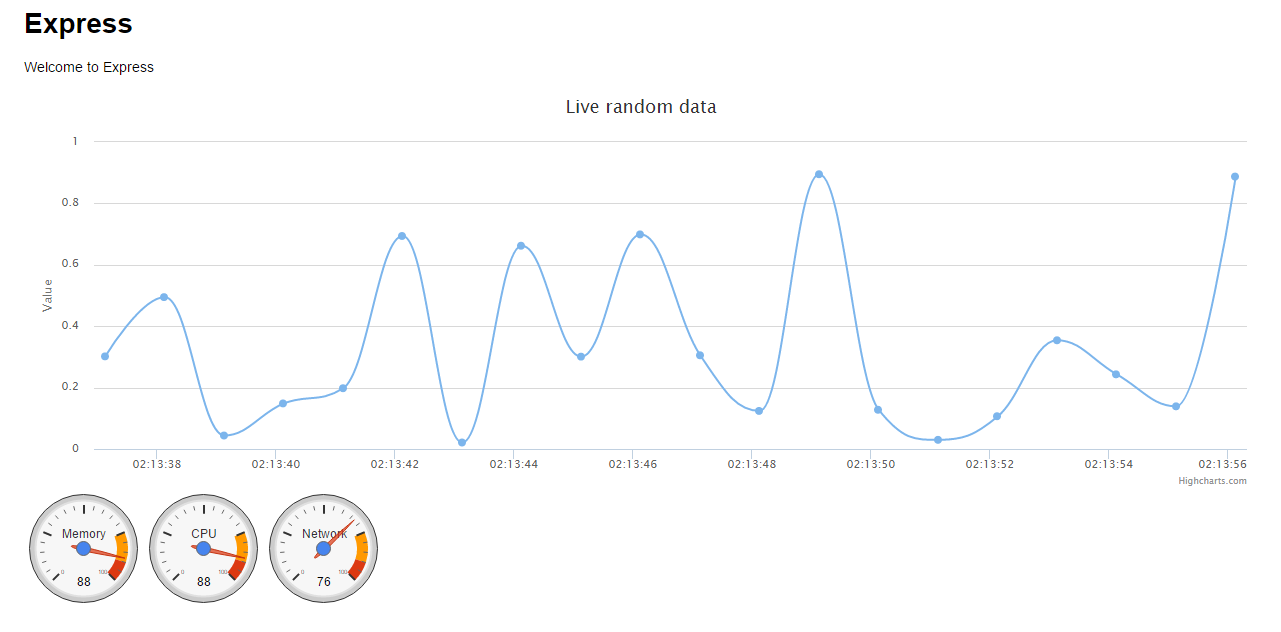
En console :



1. Partie NodeJS
2. Introduction

Pour cette deuxième partie, nous devions créer une application web récupérant les informations de notre objet et affichant ses statistiques.

Pour cela nous devions réaliser les étapes suivantes :

1. Récupérer les informations de votre application connectée à votre objet, soit *via*SocketIO, soit *via* un server TCP tournant sur NodeJS.
2. Stocker ces informations dans une base de données (SQL ou NoSQL)
3. Afficher sous forme de graphiques les statistiques de votre objet via une bibliothèque JavaScript (D3.js, ChartJS, Google Charts …).
4.  Interface graphique

Nous avons d’abord installé Highcharts. Nous avons modifier les fichier **index.jade** et **layout.jade** situé dans le dossier **views** pour réaliser et afficher le graphique.

Ensuite pour réaliser la base de données nous avons installé NodeJs, express :

npm install node.js

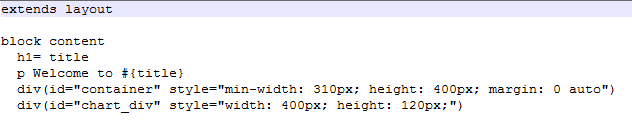
npm init

npm install express-generator -g

Puis créer deux modèles, pour le taux de CPU et le taux d’occupation mémoire **Cpu.js** et **Memoire.js** dans le dossier **models.**

layout.jade





index.jade

Nous avons créé un fichier **chart.js** dans les dossier **public -> javascripts** qui permet de gérer le graphique et les pendules.

1. Les problèmes rencontrés

Lors de ce projet, les problèmes que nous avons rencontrés sont plutôt liés à la partie en NodeJS pour récupérer les données dans la base de données puis les afficher sur le graphique ainsi que les pendules.

Pour la partie internet des objets, les modifications des paramètres de l’objet et son comportement étaient compliqués à réaliser.

1. Conclusion

Pour conclure, pour la première partie nous avons pu réaliser la majorité des fonctionnalités mais il y a plusieurs étapes que nous n’avions pas pu réaliser dans la partie NodeJS à cause des difficultés que nous avons rencontrée.