# Création d'une interface sous Qt en utilisant le noyau FreeRTOS

**Auteurs** : Benjamin ZHENG , Jonathan TONÈS
Élève Ingénieur de 1ére année
ISIMA Clermont-Ferrand

**Tuteur** : Jacques LAFFONT

**Année de réalisation** : 2016

## Remerciements

Nous remercions Monsieur Jacques LAFFONT pour avoir proposé ce projet et pour tout le temps et l'aide qu'il a mis à notre disposition pour le mener à bien.

## Introduction

L'objectif de notre projet est de créer une interface sous Qt, permettant de communiquer via le protocole TCP avec le noyau FreeRTOS. FreeRTOS est un système d'exploitation temps réel faible d'empreinte, portable et Open source pour microcontrôleur mais il est aussi disponible pour Windows.

Ce projet a été réalisé durant notre première année d'élève ingénieur de l'ISIMA sous tutelle de M.Jacques LAFFONT, professeur à Polytech.

## Déroulement du travail

Notre première approche fut d'abord de faire tourner le noyau FreeRTOS avec une démonstration basique et de mieux nous approprier le sujet.

Nous avons ensuite, fait des recherches afin de d'apprendre le fonctionnement d'un socket et également créer une interface sous QT. Une fois nos recherches effectués et nos premiers tests réalisés nous avons ensuite, chacun de notre côte appliquer nos connaissances pour qu'elles correspondent aux attentes de notre projet.

Enfin, nous avons réuni nos codes, pour ne former qu'un seul code.

Ce rapport détaille tout le travail qui a été effectué et reprend toutes les étapes qui ont été expliquées précédemment.

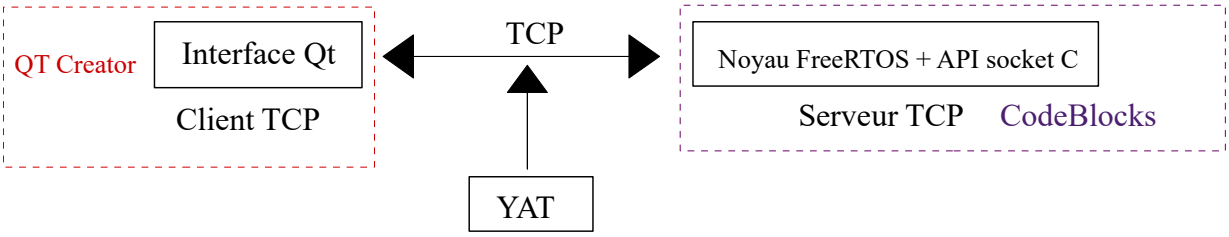
# Sommaire

# Réalisation

Afin de mener à bien notre projet, il a fallu utiliser différents outils tel que :

- Qt Creator, qui a permis la création de l'interface TCP client,
- Codeblocks, avec lequel nous faisons tourner le noyau FreeRTOS et qui a servit pour la création de l'API serveur TCP,
- YAT, qui nous sert à vérifier le fonctionnement de nos sockets. Car YAT, simule un client TCP ou un serveur TCP.

Voici le schéma récapitulatif permettant de mieux comprendre l'implication de nos outils:

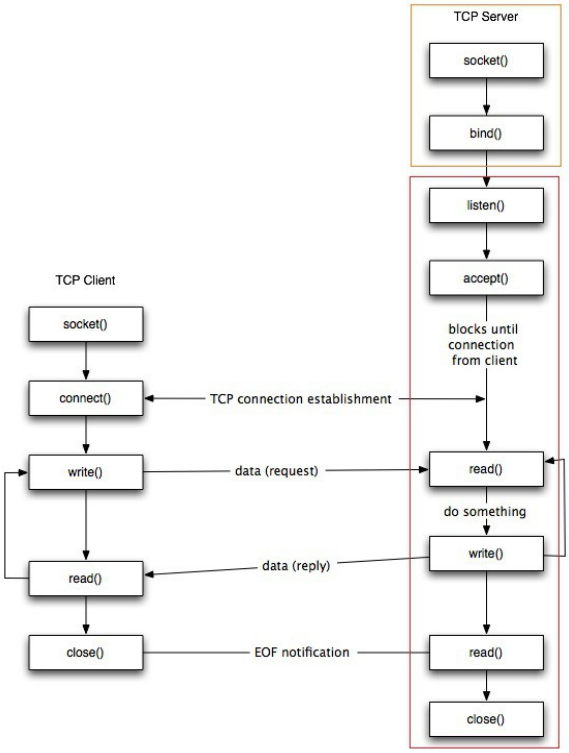



Nous avons réparti notre travail en deux parties, une partie client Qt TCP et une autre partie FreeRTOS/API socket serveur. Benjamin s'est occupé de la partie client Qt TCP et Jonathan l'autre partie FreeRTOS/API socket serveur.

# Partie C

Le code pour la programmation de l'API socket serveur a été subdivisé de la manière suivante :

Afin d'éviter d'obtenir une fonction trop grande et illisible, j'ai découpé le code l'API socket en deux fonctions. Une fonction qui s'occupe uniquement de la connexion et un autre qui charge la communication avec le client.




Ici, la partie en orange est dans une fonction appeler « `openTCPSocket` ». Dans cette fonction, il y a la vérification de toutes les variables servant à démarrer la création d'un socket. Si un problème provient, un message s'affiche permettant de connaître plus rapidement la localisation de ce problème.

Ensuite, la partie en rouge est dans une fonction appeler « `APISocket` ». Cette fonction appelle dans un premier temps la fonction `openTCPSocket` et vérifie si celle-ci n'a pas retourné d'erreur.

Dès qu'un client se connecte il y a l'affichage de son adresse IP suivi d'un message.

Tous les messages reçus du client sont stockés dans un tableau appeler « `bufferLocal` » avec une

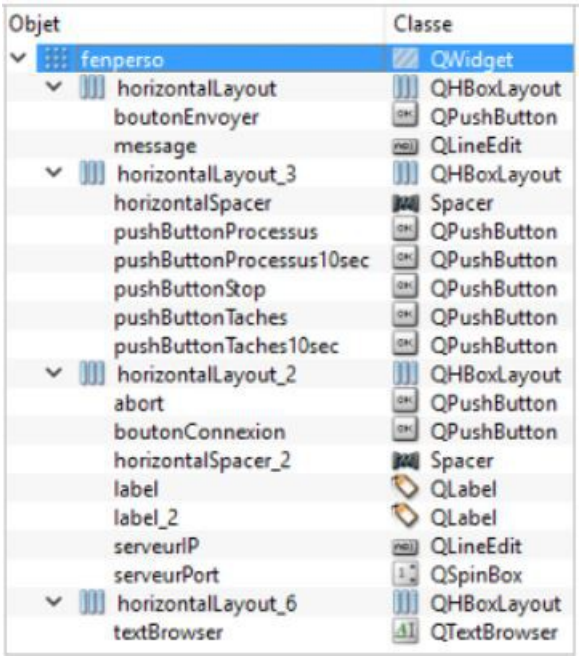
taille suffisamment grande. Si la taille pose problème, il faudra modifier la valeur définie au début dans une macro.
Un autre buffer est aussi utilisé pour stocker le résultat de la liste des tâches actives.

Pour s'assurer que le buffer ne contient pas des traces d'ancien message reçu, il est réinitialisé après chaque réception.
Lorsque le client nous envoie un message, ce message est altéré et contient des espaces entre chaque caractère. Le message est transmis dans une fonction qui se charge de remettre en formele message. Une fois le message remis en forme, les bonnes informations sont renvoyées au client.

Une boucle infinie est utilisé pour permettre plusieurs connexions successives du client et aussi afin de ne pas rester bloquer.

## Partie Qt

Tout comme la partie de l'API socket, le code de la partie QT a été découper en plusieurs fonction appeler méthode répartie en plusieurs fichiers qui sont :



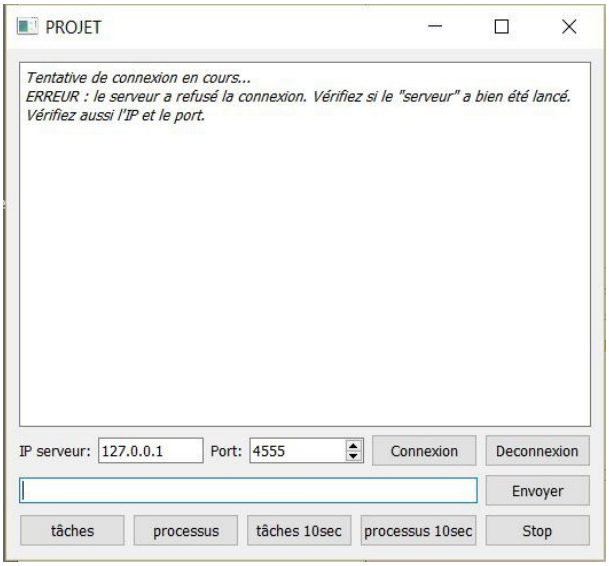

- test.pro (fichier relatif à Qt)
- fenperso.h ( fichier d'entête, contenant la déclaration des méthodes )
- fenperso.cpp ( contient le code des méthodes )
- main.cpp ( appeler des objets )
- fenperso.ui ( fenêtre )

Voici à droite l'interface sous Qt.
La partie graphique a été réalisée à l'aide de Qt Designer (*`fenperso.ui`*)

L'interface est très simple car il y a peu de boutons à avoir. Derrière chaque bouton il y a une méthode.

Cette interface permet de communiquer avec un autre serveur tout en changeant le port et adresse IP.

Le bouton `tâches` et `processus` ont été créer



pour éviter que l'utilisateur tape `tâches` et `processus` pour obtenir ces informations. Mais, l'utilisateur peut taper d'autre commande.

Aussi, l'utilisateur a la possibilité d'obtenir une évolution progressive des tâches encours avec les boutons `tâches 10sec` et `processus 10sec`. L'arrêt de ces fonctions est possible avec le bouton stop.

Les informations reçues du serveur sont affichées dans la fenêtre principale.
L'utilisateur est informé lors d'un problème de connexion ou lors de la connexion/déconnexion du serveur par le biais de cette fenêtre.

`fenperso` est lui-même de type `grid` pour permettre le redimensionnement des objets dans la fenêtre si on modifie sa taille et on utilise des `layouts` pour que la mise en forme reste consistante.

# Problèmes rencontrés

## *Partie FreeRTOS*

Les problèmes rencontrés avec le noyau FreeRTOS ont été dans l'assemblage du noyau. Il a fallu comprendre le rôle de chaque partie, pour pouvoir ensuite bien assembler et former le noyau. Néanmoins, l'assemblage du noyau n'était pas suffisant, il a fallu parfois comprendre les origines des erreurs. Souvent ces erreurs étaient des erreurs de linkage. Malheureusement, le site officiel de la documentation de FreeRTOS n'a pas averti de ces problèmes. Heureusement, quelques heures de recherches permettent de trouver la solution adéquate.

## *Partie Qt*

Qt était complètement nouveau pour moi au début, il a fallu dans un premier temps apprendre son fonctionnement. Cela est assez conséquent comparé au cours en C que nous avons eu. Par exemple les notions telles que : les objets, les signaux/slots, l'héritage, la partie graphique et réseau ont été nouvelles. Dans un second temps, il a été nécessaire de les maîtriser car le programme les utilisent comme base. De plus, la documentation fournie par Qt étant assez techniques et avec peu d'exemples, j'ai dû expérimenter avec pas mal de fonctions pour savoir exactement ce qu'elles font car les tutoriels disponibles sur internet ne traitent évidemment pas du projet sur lequel on travaille.

## *Partie TCP*

Le principal problème a été de savoir qu'est-ce que l'on reçoit ? et qu'est-ce que l'on envoi ? On a choisi d'utiliser le format suivant : [Taille du message + contenu du message]. Cependant, on s'est aperçu que les données reçues correspondaient presque à ce format. Le contenu du message était altéré, des espaces s'étaient rajoutés dans le message lors du passage Qt à C. Nous avons donc fait nos propres fonctions de décodage.

Pour passer du C à Qt, la difficulté a été de trouver le moyen de passer le tableau contenant les informations des tâches ou processus. Nous avons trouver comme solution le fait de préciser tout d'abord le mot « processus » ou « tache », et avoir ensuite les éléments du tableau, qui sont sur une seule ligne délimitée par le symbole « | ».

# Conclusion

Ce projet nous a permis d'approfondir nos connaissances en C et C++ et de découvrir de nouvelles notions. C'est également une découverte du TCP car nous n'avons jamais eu à faire de programmes communicant en réseau. Nous avons également dû tester et résoudre certaines anomalies, à l'aide de YAT et des débuggers intégrés, liées à la communication TCP bien que le code était syntaxiquement correct et bon dans le principe.