# http://www.polymtl.ca/sc/img/logoType/logoGenie/FR/gauche/polytechnique_genie_gauche_fr_rgb.pngINF3405

# Réseaux Informatiques

# Projet Final

# Par :

# Gildéric Deruette 1718065

# David Mainville 1636075

# Section de labo #1

# École Polytechnique de Montréal

# 7 Avril 2014



# Introduction

Ce projet consiste en la réalisation de deux applications réseau pouvant communiquer via sockets TCP.

L’une des applications est un serveur qui doit être en mesure d’accepter la connexion de nombreux clients, la seconde application est le client qui doit être apte à se connecter au serveur et en récupérer des informations.

Les objectifs pédagogiques de ce projet sont la familiarisation avec le concept de client et de serveur ainsi que la compréhension de la communication par sockets. Également, comme le serveur doit être en mesure de répondre à plusieurs clients, certaines notions sur les threads sont primordiales.

Pour réussir ce TP, il est essentiel de comprendre le fonctionnement des sockets sans quoi la communication entre le client et le serveur ne sera pas possible. Il faut également comprendre en surface les threads sinon le serveur ne sera pas apte à répondre à plusieurs clients en parallèle.

# Présentation

Cette section du rapport vise l’explication détaillée du fonctionnement de notre solution.

Le premier requis était que le client puisse se connecter au serveur. Ainsi la première étape lors du démarrage du serveur est de créer un socket qui écoute sur un port spécifié par notre programme (nous avons utilisé le port 5001). Une fois le serveur en attente il est possible pour des clients de se connecter à ce dernier. Par conséquent, lors du démarrage du client, la première étape est de leur demander d’entrer l’adresse du serveur afin de créer un socket client qui se connecte au socket serveur.

Une fois la connexion établie, la seconde étape consiste à informer le client des différentes options du serveur. Ainsi, le serveur transmet via le socket TCP des paquets de 100 octets contenant des strings représentant les différentes options possible. Suite à l’envoie de chaque option, le serveur envoie un dernier paquet de 100 octets contenant le code ‘FIN’ qui permet au client de comprendre que l’envoie d’options est terminé (cette dernière étape est nécessaire puisque le client n’est pas conscient de la quantité total d’option, il reste donc en mode ‘réception d’option’ jusqu’à l’obtention du message de fin).

Une fois toutes les options reçues par le client, ce dernier les affiches dans la console afin de permettre à l’utilisateur de faire un choix. Pour faire un choix l’utilisateur client doit entrer un char suivit de la touche enter. Chaque option est associée à un chiffre et chaque opération est associée à une lettre. Ainsi, dans le cas où il y aurait 7 options, l’utilisateur pourrait entrer des chiffre de 1 à 7 (qui représente une options que le client souhaite recevoir), la lettre A (pour aide), la lettre Q (pour quitter) ou la lettre E (pour envoyer). Si l’utilisateur tente d’entrer un caractère autre, le programme l’en avertira et un nouveau choix lui sera demandé. Le client peut choisir plusieurs options, chaque fois qu’il choisit un nouveau chiffre il verra ce dernier s’ajouter à sa sélection courante. S’il souhaite soustraire une option de sa sélection il n’a qu’à l’entrée de nouveau (ce fonctionnement est expliqué en détail lorsque l’utilisateur choisit le menu d’aide).

Une fois sa sélection faite, l’utilisateur client n’a qu’à entrer la lettre E pour envoyer ses choix au serveur. Le serveur lui répondra avec des données aléatoires correspondantes au choix du client. Le client pourra ensuite visionner les valeurs retournées par le serveur. À ce point, le client sera déconnecter du serveur et le thread qui l’écoute depuis le serveur sera détruit. Le client pourra donc prendre tout son temps pour observer les informations voulu après quoi il sera invité à quitter l’application.

Il est à noter que chaque opération du client peut être faite par d’autres clients en parallèle. Le serveur est apte à répondre à plusieurs client puisqu’il crée un nouveau thread d’écoute suite à chaque connexion d’un client (ces thread sont, comme dit précédemment, détruit une fois que le client a terminer ses opérations).

# Difficultés rencontrées

Le projet, dans son ensemble, s’est déroulé sans trop d’accro. Cependant, à quelques reprises nos connaissances des threads ont été mises à l’épreuve.

Également, les procédures d’utilisation des sockets ont dû être révisées puisqu’à certain moment nous avons effectués des opérations sur des sockets qui n’étaient pas connectés. Heureusement, des messages d’erreurs concernant les sockets nous ont permis de repérer rapidement ces erreurs et de les corriger (en connectant les sockets avant d’effectuer des opérations).

Finalement, un problème mineur de port nous empêchait de communiquer du client au serveur. Le problème venait fort probablement du fait que le firewall bloquait certains ports. En changeant le port pour 5001 nous avons réglé ce problème.

# Conclusion

Le projet nous apparaît comme une réussite. Il nous a permis d’approfondir grandement notre compréhension des réseaux.

L’analyse de l’architecture serveur-client était sans aucun doute fort utile.

Dans son ensemble, le projet était suffisamment difficile pour nous forcer à en apprendre d’avantage. Les quelques difficulté que nous avons rencontré furent fort pédagogiques.

C’est pour toutes ces raisons que nous considérons ce projet comme approprié, nos attentes sont comblées.