

Jérémy Barrette – 1736976

Alexis Vailles – 1742139

**Rapport TP #1 :**

**Projet en réseaux informatiques – Gestionnaire de fichiers**

Soumis à : Kadi, Mehdi

INF3405 (01 – B1) – Réseaux informatiques

Session Automne 2018

École Polytechnique de Montréal

Vendredi le 26 octobre 2018

# **Introduction**

De nos jours, le besoin d’espace de stockage numérique est de moins en moins un problème. Désormais, de nombreuses plateformes de stockage en ligne, comme Dropbox ou Google Drive, permettent de sauvegarder nos fichiers informatiques sur le web et d’y avoir accès de partout. Physiquement, ces fichiers sont toutefois stockés sur un serveur externe, auquel nos ordinateurs ont accès via une connexion.

C’est ce type de technologie que nous fait explorer le premier travail pratique (TP) du cours *INF3405 : Réseaux informatiques*. L’objectif de ce laboratoire est de programmer nous-mêmes un stockage sur un serveur externe, en programmant à la fois les fonctions nécessaires pour le compte client et le compte serveur, en ne conservant pour l’instant que l’interface console.

# **Présentation des travaux**

La première chose que nous avons faite a été d’implémenter correctement la communication entre le serveur et le client, conformément aux exigences dans l’énoncé du laboratoire. Il s’agissait, essentiellement, de s’assurer que l’adresse IP entrée était dans le bon format (respectant le nombre de bits, uniquement des nombres positifs et aucun caractère autre que des chiffres), et que le port soit compris entre 5000 et 5050. Nous avons donc imposé une série de conditions pour que le code respecte ces conditions, et qu’il rejette toute adresse entrée en dehors des plages permises.

Il était ensuite demandé d’implémenter 6 commandes : les fonctions **cd**, **ls**, **mkdir**, **upload**, **download**, et **exit**.

Pour coder la commande cd, qui permet de se déplacer vers un répertoire parent ou enfant, nous avons utilisé une variable de type Path pour identifier le chemin et un string pour lui attribuer un nom. Il s’agissait ensuite de comparer la fin de l’index avec cette position pour déterminer l’endroit où on se situe.

La commande ls, qui sert à afficher les dossiers et les fichiers dans le répertoire courant, a été implémentée avec un tableau dans lequel les positions des répertoires étaient inscrites. Nous avons ensuite créé des boucles logiques pour afficher les noms des dossiers et des fichiers dans le répertoire désiré.

La commande mkdir a pour but de créer un dossier dans le serveur à partir du client. Pour l’implémenter, nous n’avons eu qu’à utiliser la fonction *createDirectory*, en imposant les conditions nécessaires pour éviter de créer un dossier déjà exstant.

La commande upload vise à télécharger un fichier du client dans le serveur. Pour réaliser son implémentation, on commence par vérifier que le fichier à envoyer existait déjà dans le répertoire, puis ensuite on créé un Buffered Input stream du fichier sur l'envoyeur et un Buffered output stream pour le receveur afin de lire et écrire le fichier. On envoie ensuite le contenu du fichier dans un array (tableau) d’octets par le socket entre deux streams de données.

La commande download a pour but de télécharger un fichier du serveur dans l’ordinateur du client. Son implémentation est sensiblement la même que pour la commande upload, mais avec le transfert fait de manière inversée.

Finalement, la commande exit permet au client de se déconnecter du serveur. Elle a été implémentée simplement en lisant la commande « Exit » entrée par l’utilisateur et se déconnecter avec la fonction *system.exit(0)* en affichant un message à l’écran.

# **Difficultés rencontrées**

Plusieurs difficultés ont été rencontrées au cours de la complétion de ce TP. D’abord, nous avons remarqué que l’utilisation de la fonction *in.readLine()* bloquait les retours à la ligne, ce qui nous empêchait d’envoyer plusieurs lignes en même temps pour la commande Is.

Ensuite, nous avons eu de la difficulté à bien lire les adresses et leur port, puisque deux caractères coupaient la lecture : le « . » et le « : ». Nous avons remédié à ce problème en créant un tableau string nommé *parsedAddress[]* pour bien séparer les valeurs de l’adresse IP et du port.

Finalement, nous avons eu de la difficulté avec l’ajout d’un « \n » (retour de chariot) à la fin des strings, puisque cet ajout décalait tous les messages pour le client, et à l'inverse, le client attendait un message qui ne venait pas, ce qui le faisait bloquer.

# **Critiques et améliorations**

À la lumière de ce qui a été accompli au cours de ce laboratoire, nous croyons que quelques améliorations pourraient être apportées.

D’abord, pour faciliter la compréhension des étudiants, ce serait une bonne idée de séparer ce laboratoire en deux sous-laboratoires : un visant à comprendre et se familiariser avec les *sockets*, et le second visant à se concentrer uniquement à la programmation du serveur et du client.

Pour contourner ce problème du manque de connaissances par rapport aux sockets, il aurait aussi été possible de fournir un exemple de transfert de fichier par socket (autre qu’avec les fonctions *println* et *read*), pour utiliser cette méthode plus facilement.

# **Conclusion**

En conclusion, nous pouvons affirmer que ce laboratoire nous a été fort utile. Il nous a permis, entre autres, de nous initier à la communication entre serveur-client, un élément de base dans le cadre du cours sur les réseaux informatiques. Ces connaissances risquent d’être fort utiles dans nos futures carrières d’ingénieurs, et constituent une excellente base de connaissances pour la suite du cours et du programme.