
Département de génie informatique et génie logiciel

INF3405

Réseaux Informatiques

Hiver 2019

# Rapport du TP1 : Projet en réseaux informatiques

## Gestionnaire de fichier

Soumis à Émilie Dion-Paquin

Julien Legault 1847125

Abdellah Rahmani 1965352

Le 28 février 2019

**Introduction**

Le but du laboratoire était de concevoir une application client-serveur permettant à un usager de stocker des fichiers, à l'aide d'un certain nombre de commandes, sur un serveur de stockage afin d'arrêter d'encourager les géants de l'infonuagique qui limite notre espace de stockage d'une main en nous extirpant notre argent de l'autre. Il est alors notre tâche de bâtir une application client-serveur permettant à l'usager de se connecter à son espace personnel afin de téléverser un fichier sur le serveur, télécharger un fichier présent sur le serveur, supprimer un fichier présent sur le serveur, afficher tous les fichiers stockés sur le serveur ou quitter l'application terminant ainsi la communication entre le client et le serveur.

Pour faire fonctionner notre application, nous devions simuler le comportement de toute l'application client-serveur. En effet, si nous nous penchons sur la théorie. Quand nous envoyons une requête http pour nous connecter sur Facebook par exemple, l'URL est remplacée par l'adresse IP des serveurs Facebook grâce au serveur DNS. Quant à notre application nous ignorons la première partie et fournissons directement l'adresse IP du serveur. C'est pour cette raison qu'il faut que le serveur et le client connaisse l'adresse IP du serveur. De plus, nous avons besoin du port d'écoute du serveur. Ce port est placé dans chacun des paquets envoyés au serveur afin de connaitre le port de destination. Chaque client est ensuite connecté sur son propre thread et son propre port afin que le serveur puisse retourner les bonnes requêtes au bon client sans entremêler les requêtes des différends clients.

**Présentation**

Pour implémenter les différentes requêtes, nous avons d'abord commencé par établir les différentes *cases* du côté client et serveur. Ainsi, il était facile de se repérer et d'afficher les bons messages du côté client et d'appeler les bonnes méthodes du côté client et serveur. Par exemple, si nous étions dans le cas où le client venait d'insérer le bon mot de passe, nous l'invitions à entrer une commande alors que si l'usager insère le mauvais mot de passe trois fois de suite, nous le déconnectons. Pour ce qui est de la communication client-serveur, nous avons utilisé des *ObjectOutputStream* et *ObjectInputStream* pour, respectivement, envoyer et recevoir des messages. De plus, c'est avec le *ObjectInputStream* que nous pouvons déterminer dans quel cas nous sommes puisque lorsque nous envoyons un *ObjectOutputStream* nous ajoutons au début du message un *id* permettant d'identifier la *case* approprié.

Regardons à présent comment nous avons implémenté les commandes :

Tout d'abord, pour la commande *ls* permettant d'afficher le contenu de l'espace de stockage sur le serveur, nous avons une méthode qui passe au travers du dossier de l'usager du côté serveur et ajoute chaque élément au *ObjectOutputStream* qui est envoyé au client. À l'entête du message envoyé, nous ajoutons un message servant de *id* pour repérer le bon *case*. Nous devons par la suite afficher le contenu grâce à une méthode du côté client consistant à une boucle for affichant chaque item.

Ensuite, pour les commandes *Download* et *upload* permettant à l'utilisateur de télécharger un fichier à partir du serveur ou de transférer un fichier vers ce dernier respectivement, elles sont assurées par la méthode *load* qui commence par vérifier l'existence du fichier à télécharger ou à téléverser, pour ensuite procéder à l'opération désirée via la méthode *copyFile*. La commande désirée est identifiée par un *id* permettant d'indiquer le type de la commande *Download* ou *upload.* La méthode *copyFile* permettra de créer une copie du fichier à télécharger ou à téléverser dans le répertoire de destination (répertoire local ou répertoire sur le serveur). Des messages indiquant le statut de la commande (fichier téléchargé avec succès, fichier inexistant, etc.) sont envoyés au client via un *ObjectOutputStream.*

La commande *Delete* qui permet de supprimer un fichier depuis le répertoire de l'utilisateur sur le serveur, accède à ce dernier et supprime le fichier en question s'il existe, puis transmet un message au client indiquant soi la suppression du fichier ou non (dans le cas où il s'agit d'un fichier inexistant par exemple)

Et enfin, pour implémenter la commande exit, si nous nous retrouvons dans ce *case,* nous appelons une méthode qui va, d'abord, envoyer un message au client pour récupérer le case déconnection du côté client puis va fermer le socket dans un *try block* afin d'attraper les exceptions.

**Difficultés rencontrées, critiques et améliorations**

Lors du travail, nous avons rencontré un problème au niveau de l'utilisation des *ObjectOutputStream* et *ObjectInputStream* utilisés afin d'établir la communication entre le client et le serveur. Le problème rencontré est que lors de la création d'un objet de type *ObjectInputStream* le programme se bloque jusqu'à ce qu'un objet soit lu. Cependant, comme un Objet était créé du côté serveur et client, on se retrouvait rapidement en situation d'interblocage ou le serveur attend un objet du client et vis versa. La solution élaborée pour contourner le problème a été de créer une première méthode ne faisant que prendre une *ObjectOutputStream* afin d'envoyer

le nom d'utilisateur, du côté serveur, la méthode traitant toutes les requêtes avec le *ObjectInputStream et le ObjectOutputStream* est appelée pouvant ainsi se débloquer pour recevoir l'objet. Cette méthode envoi ensuite un objet au client et se bloque ce qui débloque la méthode principale du côté client traitant toutes les requêtes. Au niveau de notre critique du travail, nous avons trouvé ce laboratoire très intéressant dû au fait qu'il s'agit d'un laboratoire où nous commençons avec les mains vides et il faut tout développer pour arriver avec une application très intéressante. Cela fait en sorte que nous gardons une motivation tout au long du TP et cherchons constamment à améliorer le travail.

**Conclusion**

Ce travail pratique était l'occasion pour nous familiariser avec les applications client-serveur, en effet; nous avons pu voir de manière concrète les échanges de données entre le client et le serveur en utilisant les *sockets*, c'était donc une mise pratique des notions déjà vu en cours.

Aussi, le fait de concevoir l'application du début jusqu'à la fin en partant du cahier de charge résumant l'ensemble des requis attendus, nous a permis d'améliorer notre capacité à analyser les besoins fonctionnels et non fonctionnels de l'application client-serveur et d'y apporter les solutions simples et efficaces

**Ressources utilisées**

- Les codes sources permettant la communication simple client-serveur fournies par les chargés de laboratoire.
- Des exemples des java sockets disponibles sur le lien : http://cs.lmu.edu/~ray/notes/javanetexamples/
- Utilisation des Regex pour la validation d'une adresse IP (titre : 2.Using OWASP Validation Regex) disponible sur le lien : https://www.techiedelight.com/validate-ip-address-java/