**INF3405 – Réseaux Informatiques**

**Hiver 2023**

**TP No. 1 :**

**Projet en réseaux informatiques**

**Gestionnaire de fichier**

**Groupe 02, 04, 05**

**2153068 – Selamet Zamanoglu**

**2148614 – Ilias Bakhbukh**

**2145194 – Mohamed Chaoui**

**Soumis à : Bilal Itani, Mehdi Kadi**

**17 février 2023**

**Introduction :**

Il y a deux objectifs principaux pour ce laboratoire. Il permet à l’étudiant de se familiariser avec l’utilisation des sockets pour la communication client/serveur et d’utiliser les threads pour le développement des applications réseau. Ces objectifs permettent aussi l’évaluation des qualités 4 (conception) et 7 (communication) d’un étudiant en ingénierie. L’étudiant est placé dans le contexte qu’il est fatigué de ne pas avoir suffisamment d'espace de stockage sur les plateformes de stockage cloud courantes, décide de créer sa propre application client-serveur. Il utilise un vieil ordinateur donné par sa grand-mère comme serveur de stockage et se concentre sur le développement de l'application plutôt que sur l'interface graphique.

**Présentation :**

1. Saisie des paramètres du serveur dans l’interface console du client et celle du serveur (adresse IP et port d’écoute entre 5000 et 5050).

On a utilisé un « Scanner input » pour la saisie des paramètres dans l’interface console. Ensuite, on appelle nos fonctions « readAdress et readPort », qui sont implémentés dans notre classe « Tools », qui prennent l’input en question comme argument.

2. Vérifier la validité de l’adresse IP saisie (uniquement le format) et le numéro de port (entre 5000 et 5050)

On a utilisé la librairie native de Java intitulé « regex » pour la validation d’une adresse IP et le numéro de port dans nos fonctions « ipValidation et portValidation ». On a assigné le pattern voulue (selon l’adresse IP ou le numéro de port) dans un string et ensuite appelé « Pattern.compile et Pattern.matcher » pour valider une saisie selon ce pattern. Nos fonctions « readAdress » et « readPort » appelent nos fonctions de validations et demande un nouvel input du client jusqu’à ce qu’il saisit une adresse IP ou un port d’écoute valident.

Pour les commandes, on a utilisé une boucle do while dont la condition est que la commande écrite par le client n’est pas « exit ». À l’intérieur de cette boucle, on prend l’input du client avec le scanner initialisé au tout début dans le code du client. On appele notre fonction « readCommand » qui prend l’input « raw » et le sépare ou non selon le type de commande reçu. Cette fonction retourne un tableau de string qui contient 2 parties. La première partie correspond au nom de la commande (ls, cd, mkdir, …) et la deuxième correspond aux options d’une commande (le répertoire qu’on veut s’y déplacer ou créer, …). Par exemple, si la commande était un « ls », alors la première partie reste un ls, mais la deuxième est un string vide. Si c’était un « cd [folder] » alors on les sépare en deux. La classe client envoit cette commande via « out.writeUTF » pour que le « ClientHandler » puisse lire et effectuer la commande en question en appelant la fonction en question avec un switch case.

3. Téléverser et télécharger un fichier

Pour le téléversement, on appelle la méthode « upload » qui prend en entrée une chaîne de caractères "nameAndFormat" représentant le nom et le format du fichier à écrire, ainsi qu'un objet "DataInputStream" qui est utilisé pour lire les données d'entrée. La méthode crée un nouvel objet "File" à partir du nom de fichier et du répertoire courant, puis ouvre un flux de sortie pour écrire les données dans le fichier. La méthode utilise une boucle "while" pour lire les données d'entrée à partir du flux de données, puis écrit les données lues dans le fichier. La boucle s'arrête lorsque la fin du flux est atteinte ou si une erreur de délai d'attente se produit. Enfin, la méthode ferme le flux de sortie et renvoie "true" si l'opération s'est déroulée avec succès ou "false" s'il y a eu une erreur.

Pour le téléchargement, …

4. Pouvoir se déplacer dans la hiérarchie des répertoires du serveur de stockage à partir du client

On appelle la fonction « cd » avec l’argument « commandOption qui correspond à la 2e partie d’un input du client. Elle réassigne une variable File qui s’appelle « currentFile » à un nouveau File. On ajoute au chemin de l’ancien File la partie « commandOption ». Pour finir, on affiche au client le message « "Vous êtes dans le dossier "+currentFile » qui montre le chemin absolue du répertoire courant.

5. Pouvoir énumérer les répertoires et les fichiers au niveau du serveur de stockage à partir du client

On appelle la fonction « ls » utilise la méthode "listFiles()" sur la variable « currentFile » pour obtenir un tableau des fichiers et dossiers, et utilise une boucle "for" pour parcourir le tableau. Pour chaque élément du tableau, la méthode utilise "isDirectory()" et "isFile()" pour déterminer s'il s'agit d'un dossier ou d'un fichier, puis ajoute la chaîne appropriée à la chaîne de caractères "returnString". La méthode renvoie finalement "returnString" contenant la liste des fichiers et dossiers.

6. Pouvoir créer un répertoire à partir du client sur le serveur de stockage à partir du client

On appelle la fonction « mkdir » (la nôtre) qui prend comme argument "commandOption" représentant le nom du dossier à créer. On crée un nouvel objet « File » en utilisant le répertoire courant et commandOption. La fonction appelle la méthode native de java « mkdir » pour créer le dossier voulu et retourne un booléen selon le succès de l’opération.

7. Pouvoir se déconnecter adéquatement du serveur de stockage

Après que le client fait une commande « exit », le code sort de la boucle « do while » et on appele « input.close() » pour fermer le scanner des saisies du client dans l’interface de la console et « socket.close() » pour fermer le socket qui permet la communication entre un client et le serveur.

8. Afficher en temps réel les demandes à traiter (logs au niveau de la console serveur

À chaque demande à traiter, on appelle la fonction «System.out.println » pour les afficher, mais le client envoit les commandes au serveur avec la fonction « out.writeUTF ».

**Difficultés rencontrées :**

1. Envoyer une image 4K était compliqué, car on ne pouvait pas envoyer toute l’image en 1 coup. On a dû envoyer l’image en plusieurs paquets de 8kb.
2. On a eu de la difficulté par rapport à la duplication de code et de lisibilité. On a du créé une classe « Tools » pour séparé en fonctions les solutions pour le tp.

**Critiques et améliorations :**

On n’a pas vraiment de critiques et améliorations par rapport au laboratoire. Nous avons trouvé ce laboratoire assez utile et intéressant à faire.

**Conclusion :**

Ce laboratoire sort des tp ordinaires auxquels ont été habitué. On apprit à faire communique un client et un serveur grâce à la réseautique avec les socket. Nos attentes ont été comblés.