École Polytechnique de Montréal

Département Génie Informatique et Génie Logiciel

Cours : INF3405 – Réseaux Informatiques

Session : Automne 2025

Projet : TP1 - Gestionnaire de fichier Client-Serveur

Date de remise : 22 octobre 2025

Équipe :

- **Saad Jad** - Matricule : **2416127**
- Nom Étudiante 2 - Matricule :
- Nom Étudiante 3 - Matricule :

INTRODUCTION

Ce projet consiste à développer une application client-serveur permettant de gérer des fichiers sur un serveur centralisé, accessible à distance. L'objectif principal est de comprendre et d'appliquer les concepts fondamentaux des réseaux informatiques, notamment l'architecture client-serveur, l'utilisation des sockets, et la gestion de la concurrence avec les threads.

Dans le contexte d'une utilisation limitée de l'espace de stockage cloud, nous avons développé notre propre solution de gestion de fichiers en réseau. Cette application permet aux clients de se connecter à un serveur, d'explorer une hiérarchie de répertoires, et de transférer des fichiers de manière sécurisée.

Les principales technologies utilisées sont Java pour la programmation, les sockets TCP pour la communication réseau, et les threads pour supporter plusieurs clients simultanément.

PRÉSENTATION DE LA SOLUTION

Architecture générale

Notre application suit le modèle client-serveur classique avec une communication basée sur les sockets TCP. Le serveur écoute sur une adresse IP et un port spécifiés (entre 5000 et 5050), tandis que les clients se connectent au serveur pour effectuer des opérations sur les fichiers.

L'application est composée de trois classes principales :

- Server.java : Démarre le serveur, valide les paramètres (IP, port) et accepte les connexions
- ClientHandler.java : Traite les demandes de chaque client dans un thread séparé
- Client.java : Interface pour l'utilisateur, valide les paramètres et communique avec le serveur

Implémentation des commandes

Nous avons implémenté les sept commandes requises :

1. ls : Liste les fichiers et dossiers du répertoire courant avec les préfixes [Folder] et [File]
2. cd : Change le répertoire courant, supporte l'accès au répertoire parent avec ".."
3. mkdir : Crée un nouveau dossier dans le répertoire courant
4. delete : Supprime récursivement un fichier ou dossier
5. upload : Transfère un fichier du client vers le serveur avec vérification d'intégrité via MD5
6. download : Transfère un fichier du serveur vers le client

7. exit : Ferme la connexion proprement

Gestion de la concurrence

Le serveur utilise des threads pour supporter plusieurs clients simultanément. Chaque nouvelle connexion client crée un nouveau thread via la classe ClientHandler, permettant à plusieurs clients de travailler indépendamment. Chaque client maintient son propre répertoire courant, évitant les conflits.

Sécurité - Prévention du path traversal

Nous avons implémenté la méthode `secureResolve()` qui empêche les clients de sortir du répertoire racine (sandbox) en utilisant la méthode `startsWith()` pour vérifier que le chemin résolvé reste dans le répertoire de base.

Validations

Les validations suivantes ont été implémentées :

- IP serveur : Vérifie le format (4 octets, 0-255 chacun)
- Port serveur : Vérifie que le port est entre 5000 et 5050
- IP client : Même validation que le serveur
- Port client : Même validation que le serveur

Format des logs

Les logs du serveur suivent le format spécifié :

[IP_CLIENT:PORT_CLIENT - YYYY-MM-DD@HH:MM:SS] : COMMANDE

Exemple :

[127.0.0.1:55555 - 2025-10-19@14:30:45] : CONNECT
[127.0.0.1:55555 - 2025-10-19@14:30:50] : ls
[127.0.0.1:55555 - 2025-10-19@14:31:00] : upload fichier.txt

Transfert de fichiers

Pour les transferts (upload/download), nous avons implémenté un protocole robuste :

- Envoi de la taille du fichier (long)
- Transfert par buffers de 64KB pour optimiser la performance
- Vérification d'intégrité possible avec MD5 après le transfert

DIFFICULTÉS RENCONTRÉES ET SOLUTIONS

Difficulté 1 : Gestion simultanée de plusieurs clients

Problème : Notre première approche utilise une boucle while simple, ce qui bloquait le serveur lors de la connexion d'un client.

Solution : Nous avons utilisé les threads pour traiter chaque client dans un fil d'exécution séparé, permettant au serveur d'accepter les connexions suivantes sans interruption.

Difficulté 2 : Transfert de fichiers binaires

Problème : Utiliser `readUTF()` et `writeUTF()` ne fonctionne pas bien pour les fichiers binaires volumineux.

Solution : Nous avons implémenté un transfert basé sur des buffers de bytes bruts, en envoyant d'abord la taille du fichier, puis le contenu par chunks de 64KB.

Difficulté 3 : Sécurité du chemin d'accès

Problème : Un client pouvait potentiellement accéder à des fichiers en dehors du répertoire racine en utilisant des chemins relatifs complexes (../../etc/passwd).

Solution : Nous avons utilisé `Path.normalize()` et `Path.startsWith()` pour vérifier que le chemin résolvé reste toujours dans le répertoire sandbox.

Difficulté 4 : Répertoire courant par client

Problème : Partager un répertoire courant global posait des problèmes de concurrence.

Solution : Chaque instance de ClientHandler maintient son propre objet Path `currentDir`, isolant complètement l'état de chaque client.

CRITIQUES ET AMÉLIORATIONS

Points forts

- Architecture modulaire et facile à maintenir
- Gestion complète de la concurrence sans deadlocks
- Protocole de transfert robuste pour fichiers volumineux
- Sécurité adéquate contre les tentatives de sortie du sandbox

Améliorations possibles

1. Interface graphique : Une interface GUI rendrait l'application plus conviviale qu'une console
2. Authentification : Ajouter un système de login/password pour sécuriser l'accès au serveur
3. Chiffrement : Implémenter SSL/TLS pour chiffrer les communications

4. Compression : Compresser les fichiers lors du transfert pour économiser la bande passante
5. Limite de débit : Implémenter un throttling pour éviter la congestion réseau
6. Base de données : Stocker les métadonnées des fichiers dans une base de données
7. Gestion des permissions : Implémenter un système de droits d'accès par utilisateur

CONCLUSION

Ce projet nous a permis d'approfondir notre compréhension des réseaux informatiques, particulièrement en ce qui concerne l'architecture client-serveur, la communication par sockets, et la programmation concurrente en Java.

Les défis rencontrés, notamment la gestion de plusieurs clients simultanés et le transfert sécurisé de fichiers, nous ont poussés à concevoir des solutions robustes et scalables. Nous avons appris l'importance de valider les entrées utilisateur, de gérer les ressources correctement, et de prévoir les cas d'erreur.

Ce travail pratique a consolidé nos apprentissages théoriques et nous a donnés une expérience concrète du développement d'applications réseau. Nous sommes satisfaits du résultat et des compétences acquises, qui nous seront utiles pour les projets futurs en ingénierie logiciel et réseautique.