INF3405- Réseaux Informatiques

Automne 2019

Rapport

Architecture client-serveur

Présenté à

Bilal Itani

Par

Myriam Kiriakos 1888929

Huyen Trang Dinh 1846776

gr. 02

École Polytechnique de Montréal

21 octobre 2019

Introduction

Dans le cadre du cours de réseaux informatiques à Polytechnique Montréal, nous devons nous servir d'un Pentium 3 afin qu'il puisse accueillir un serveur de stockage de fichiers. À partir des connaissances que nous avons acquises lors du cours, nous avons conçu une architecture client-serveur qui permet au client de téléverser ses fichiers aux serveur et de télécharger des fichiers sur le serveur. Le client devra également être en mesure de créer des répertoires, se déplacer dans les répertoires et énumérer les fichiers dans le répertoire courant. Le serveur devra pouvoir accueillir et traiter les demandes de un plusieurs clients à la fois tout en affichant ces demandes en temps réel. En ce qui concerne la connexion entre ces deux partis, une adresse IP et un port devront être fournis par chacun afin d'initier la connexion. Ces paramètres feront objet d'une validation. Nous avons procédé avec Java, un langage possédant des librairies pertinentes pouvant faciliter l'écriture du code.

Présentation

Pour la communication serveur-client, il a fallu utiliser des sockets. En Java, il existe des librairies qui permettent la gestion facile de ces sockets. Comme le serveur devait pouvoir accueillir plusieurs clients en même temps, nous avons également multi-threadé notre serveur.

En ce qui a lieu des commandes que nous devons coder(ls, cd, mkdir, upload, download, exit), la librairie qui nous a le plus servi est java.io.File. Celle-ci nous a grandement facilité la tâche lors de la gestion des fichiers et des dossiers, car la majorité des fonctionnalités que nous devions implémenter l'était déjà dans cette librairie, soit complètement soit partiellement. Par exemple, la librairie contient une fonction nommé mkdir qui crée un dossier, tout comme la fonctionnalité mkdir que nous devions implémenter.

Puis, aucune configuration particulière n'a été utilisé dans notre projet. De plus, comme nous avons utilisé les variable d'environnement lors de la création des chemins lors des fonctions comme cd à la place de "hardcode" le séparateur dans le chemin, le code peut théoriquement fonctionner indépendamment de l'OS. Il pourrait fonctionner sur Windows ou sur Linux.

Les fonctionnalités du côté serveur sont appelée à partir d'un switch case, qui selon la commande utilisée, appelle une fonction différente. Tout ce que l'usager écrit est directement envoyé au serveur. C'est au serveur de gérer ce qu'il va faire de l'information qui lui est fourni. Autrement dit, l'usager peut envoyer n'importe quoi au serveur mais il ne recevra rien d'intéressant en retour.

Voici une démonstration proposée à exécuter du côté client de quelques capacités de notre architecture client-serveur.

```
java -jar Client.jar
java -jar Server.jar
mkdir folder
ls folder
upload uploadthis.png
ls
cd ..
download downloadthis.txt
cd folder
upload downloadthis.txt
ls
exit
```

Nous constatons après cette démonstration que la machine client contient downloadthis.txt dans son répertoire courant. Du côté Serveur, nous voyons que tout ce qui était supposé être téléversé sont présents dans les bons répertoires.

Difficultés rencontrées

La première difficulté que nous avons rencontrée se retrouvait au niveau de la communication entre le serveur et le client. Un code modèle était fourni. Cependant, nous avons dû comprendre ce que le code faisait. Nous devions nous rendre compte que les lectures sont bloquantes et qu'ils étaient en attente d'un écriture et un flush dans le tampon.

La deuxième difficulté à laquelle nous avions fait face était lors de l'écriture du code pour le téléversement et téléchargement. L'idée était d'envoyer une requête de téléchargement, par exemple, au serveur. Le client devait connaître la taille du fichier à télécharger, mais seul le serveur connaît cette taille. Le serveur doit donc renvoyer la taille afin que le client puisse procéder à s'apprêter à recevoir un autre envoi du serveur (les octets du fichier). Il était de même pour le téléversement. Ces deux fonctionnalités étaient difficiles et mêlantes à implémenter puisqu'elles requièrent plus qu'une seule requête et une seule réponse, contrairement aux autres fonctionnalités. Ce va et vient de transmission de données ont souvent brisé notre code et le débogage s'est avéré fastidieux. Nous avions beaucoup de duplication de code. Tout cela s'est résolu en réfléchissant à une meilleure architecture.

Critiques et améliorations

Nous croyons que les exigences inscrites dans l'énoncé devraient être soit plus détaillées et claires ou que le travail devrait être évalué strictement et minimalement selon les critères inscrits. Le fait de manquer certains détails sur l'implémentation, par exemple le lieu de destination des téléversement ou téléchargements, nous a fait perdre beaucoup de temps. Aussi, il n'était pas clair quelles étaient les exigences sur la qualité du code. Cette indication nous permettrait de savoir sur quoi étendre notre temps. De plus, la gestion des erreurs n'était pas spécifier. De nous montrer la façon dont le programme sera évalué nous aurait certainement aidées à organiser nos tâches.

Conclusion

En conclusion, ce laboratoire nous a été utile car nous avions appris à utiliser les File, Paths, Socket et Threads en Java. Nous nous sommes aperçues que l'un des avantages de Java est sa présence de nombreuses librairies qui permettent d'accomplir des fonctionnalités avec très peu de code. Il était très intéressant de réfléchir sur la façon dont les communications entre le client et le serveur devaient se faire. Il était un défi de décider lequel s'occupe de valider quel aspect du logiciel, par exemple. Nous avons aussi beaucoup appris sur comment la transmission de fichiers fonctionnait grâce aux sockets. En résumé, les notions que nous avons trouvées les plus importantes étaient celles concernant les sockets et les fils d'exécution.