Projet MultiThreads – Serveur FTP

*Louis Rabinovitch  
Maxime Merigeaux  
Manuel Malou  
Julien Riffard  
Quentin Villegente*

EPSI Bordeaux

I4 2014-2015

**Table des matières**

|  |  |
| --- | --- |
| **I - Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .** | **2** |
| **II - Les attentes du projet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .** | **2** |
| **III - Equipe . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .** | **3** |
| **IV - Organisation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .** | **3** |
| **V - Réalisations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .** | **4** |
| **VI - Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .** | **5** |

**I – Introduction**

Dans le cadre de notre 4ème année d’ingénierie au sein de l’EPSI Bordeaux, il nous est demandé de mener à bien un projet ayant pour objectif de programmer avec le langage que l’on veut un serveur FTP multi-threadé permettant coté client d’envoyer et de recevoir des fichiers tout en suivant l’avancée des transferts et coté serveur de prendre en charge automatiquement les demandes du client.  
  
Concernant nos choix de technologies, le client sera développé en C# (WPF) et le serveur en C#.  
Nous présenterons tout d’abord les exigences du projet et la façon dont nous nous sommes organisés, et nous verrons ensuite les réalisations par rapport à ces attentes.

**II – Les attentes du projet**

L’important lors de ce projet est de manipuler correctement les Threads pour implémenter les fonctionnalités nécessaires au serveur FTP, et de remettre un travail propre et pertinent.

Contraintes client à respecter :

* Page de paramétrage.
* Possibilité de parcourir les disques locaux ou distants et de sélectionner un ou plusieurs fichiers.
* Possibilité de suivre l’avancée d’un transfert.
* Possibilité d’annuler un transfert.

Contraintes serveur à respecter :

* Afficher la liste des connexions avec leurs caractéristiques.
* Possibilité de clore une connexion avec un poste client.
* Remarque : L’interface du serveur est laissée à notre convenance.

**III – Equipe**

L’équipe se compose de 5 développeurs tous issus de l'EPSI Bordeaux et actuellement en Ingénierie 4 :

* Maxime Merigeaux, titulaire d’un BTS SIO obtenu à Bissy.
* Louis Rabinovitch, en cycle ingénieur à l’EPSI Bordeaux.
* Manuel Malou, en cycle ingénieur à l’EPSI Bordeaux.
* Julien Riffard, titulaire d’un BTS SIO obtenu à l’EPSI Bordeaux.
* Quentin Villegente, en cycle ingénieur à l’EPSI Bordeaux

**IV – Organisation**

Le travail a été réparti équitablement entre nous et de façon à toujours se retrouver à travailler à plusieurs sur les différentes tâches (par exemple, les côtés client et serveur ont été développés en parallèle). Nous étions ensemble lors du développement pour l’intégralité du projet. Cela nous a permis de communiquer et d’échanger entre nous aisément. Nous avons utilisé GitHub pour versionner nos sources. Nous essayons de pratiquer le Clean Code, pour qu’il soit compréhensible facilement, mais sans le surcharger.

.  
 **V – Réalisations**

**Client**  
Fonctionnellement, il est possible de sélectionner des fichiers sur le disque local pour les envoyer sur le serveur ou de sélectionner des fichiers distants pour les recevoir en local. Que ce soit pour l’envoi ou pour la réception, le client suit l’avancée du transfert au fur et à mesure, et peut annuler un transfert en cours. De plus, une commande est gérée par un thread et le nombre de threads actifs est limité à 10. Ainsi, en cas de nombre important, les commandes suivantes sont placées en file d’attente. La création de dossier et la suppression de fichier ont aussi été gérées.

Visuellement, nous avons pris pour exemple l’interface du client FTP FileZilla, qui correspond bien au client que nous devons faire. Nous avons élaboré en XAML une interface similaire à cette dernière.

L’interface permet de :

* S’authentifier
* Visualiser/naviguer dans les dossiers local/distant
* Visualiser les informations du fichier que l’on a sélectionné
* Visualiser l’état des transferts en cours
* Visualiser les traces des commandes passées

**Serveur**   
Le serveur de son côté prend en charge les demandes des clients connectés :

* Authentification
* Upload de fichier
* Download de fichier
* Suppression de fichier
* Création de dossier
* Suppression de dossier
* Presque toutes les commandes FTP sont gérées, sauf NLST, SITE, STAT, HELP, SMNT, REST, ABOR.

Il y a un thread par commande, un thread par connexion, le tout dans des threadpools séparés.  
Le serveur peut recevoir des commandes en cours d’exécution. Ces commandes sont les suivantes :

* adduser –u {username} –p {mdp} : permet l’ajout d’utilisateur : les utilisateurs sont gérés grâce à un fichier XML.
* dispose –u {username} : le thread de connexion du client concerné est tué, il est déconnecté du serveur.
* list : permet de lister les connexions clientes en affichant leur adresse IP.
* quit : permet d’arrêter le serveur.
* log : permet d’afficher les logs : toutes les actions intervenues sur le serveur.
* help : permet d’afficher la liste des commandes.

**VI – Conclusion**

Pour conclure, nous avons remplis les objectifs que nous nous sommes fixés au début, c’est-à-dire réaliser une interaction client/serveur multi-thread permettant de download ou d’upload grâce à l’interface client que nous avons mis à disposition, et gérer les connexions clientes coté serveur.   
  
Ce projet nous a permis de monter en compétences d’une part sur le multi-threading, qui était la problématique centrale de notre développement, et d’autre part sur les différents langages que nous avons choisis.

Cela nous a permis d’enrichir et d’améliorer nos compétences en travail de groupe, grâce à une organisation structurée et efficace, ainsi qu’une répartition du travail réfléchie.