Master 1 ALMA
Université de Nantes
2011-2012

Projet Réseau
Réalisation d'un serveur FTP

MARGUERITE Alain
RINCE Romain

Université de Nantes
2 rue de la Houssinière,
BP92208, F-44322 Nantes cedex 03, FRANCE

Encadrant : Pierrick PASSARD

Table des matières

1 Introduction

1.1 Objectifs et choix

Dans le cadre de l'apprentissage des fonctionnements des interfaces systèmes/réseaux et des sockets, nous avons choisi de réaliser un serveur FTP. Le programme sera réalisé en C. Les manipulations des sockets se font donc au plus bas niveau. Cette absence d'abstraction nous a obligé à comprendre les différents comportements des sockets. Le choix de réaliser un serveur FTP a été conditionné par les raisons suivantes

- L'opportunité de progresser dans la compréhension du fonctionnement des application clients/serveurs. En effet ce protocole est illustation simple de ce type de fonctionnement (cf 1.2).
- La relative simplicité et l'absence d'interface graphique pour laisser tout le temps de travail à la manipulations d'outils réseaux.
- Un protocole dédié au transferts de fichier est idéal pour se être confronté à un maximum de problèmes de type réseau.

1.2 Rappels sur le protocole FTP

Le protocole FTP à une architecture client/serveur spécifique dont les grandes lignes sont illustrées ci-après :

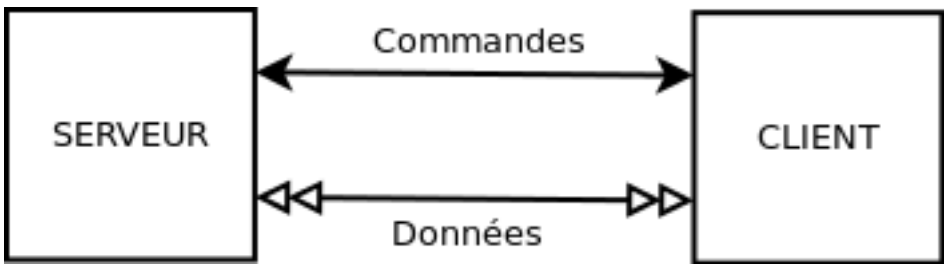
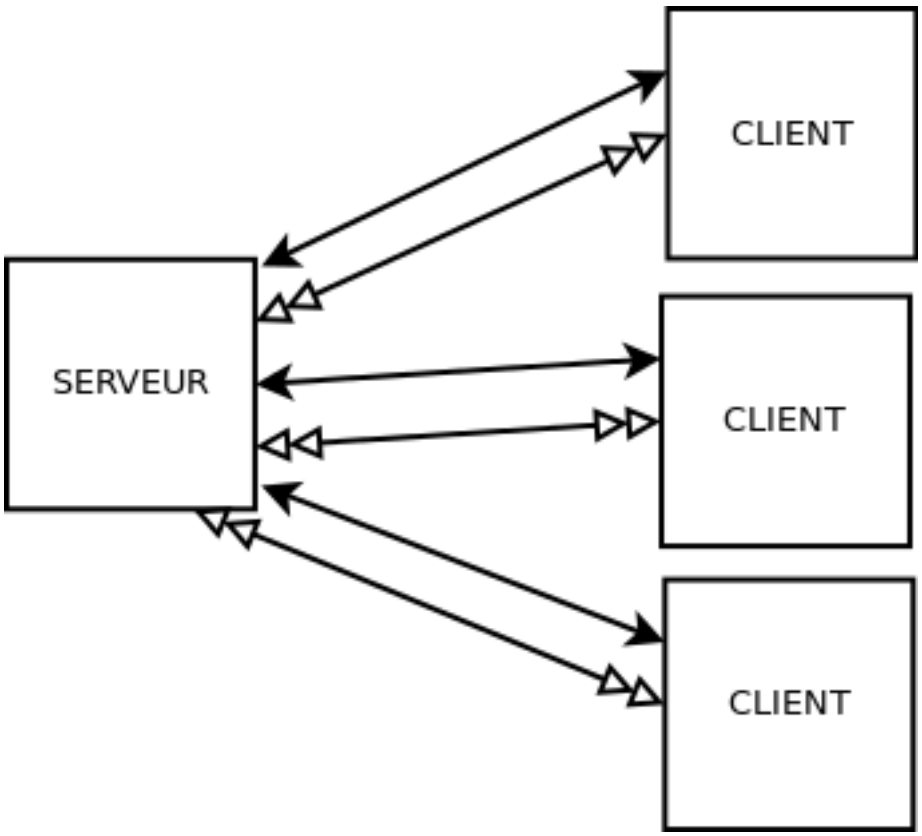


FIGURE 1.1 – Liens Serveur/Client

Deux liens de communication relie un client au serveur. Un pour un dialogue (commandes clients, informations serveurs ...), l'autre pour le transit des données. On rappel que les objectifs du serveur FTP sont les suivants :

- permettre un partage de fichiers entre machines distantes.
- permettre une indépendance aux systèmes de fichiers des machines clientes et serveur.
- permettre de transférer des données de manière efficace.

Le but de ce projet est de réaliser une telle architecture de manière à ce que plusieurs clients puissent se connecter en même temps.



2 Application réalisée

2.1 Description de l'application

2.1.1 Fonctionnalités réalisées

Les programmes qui sont ici proposé réalise les opération suivantes.

- Une connexion entre un serveur et des clients.
- Chaque client connecté se voit attribué un dossier propre au nom de son PID.
- Chaque client peut connaître le contenu du serveur, ainsi qu'opérer des commandes de la nature que $ ls sur son dossier courant.
- Chaque client peut «downloader» et «uploader» des fichiers présent dans le dossier du serveur.
  - Dans le cas d'un download, le nouveau fichier à le même nom que le fichier source
  - Dans le cas d'un upload, le nouveau fichier sera de la forme out *nomfichier*.

2.1.2 Idées d'évolutions

2.2 Structure du programme

Le client et le serveur ont chacun un programme spécifique. Ils ont cependant tous les deux accès au fonctions définies dans les fichiers network et gestionmenu. Les fonctions proposées par les fichiers network sont les plus importantes. Elle permettent entre autres d'établir la connexion entre le client et le serveur. C'est aussi là que se trouvent les fonctions de fragmentation et de fusion des différents messages. Les fichiers gestionmenu contiennent toutes les fonctions nécessaires mais qui n'ont pas de rôles pour les transferts de données (routine du menu, gestion des dossiers ...). Certaines de ces fonctions reprennent celles des fichiers network.

3 Tests et analyses

3.1 Introduction

3.1.1 Méthodes utilisées

Obtenir des un relevé temporel d'une opération de transfert s'avère problématique. En effet, il s'agit en théorie de démarrer un chronomètre à l'instant $t1$ où l'utilisateur envoie sa commande de transfert et l'arrêter à l'instant $t2$ où le nouveau fichier local au client est crée et intègre. Pour répondre à cette théorie, il aurait alors fallu transmetteur l'état de ce chronomètre entre le client et le serveur lors de cet aller retour. Cependant tous les moyens mis en place pour tenter de répondre à ce problème (transmission de la valeur du chronomètre, envois de signaux ...) prendraient eux même du temps et ainsi fausseraient le résultat. Nous avons choisi une méthode qui semblait être un bon compromis entre la fiabilité des mesures et un minimum d'effets de bords sur l'opération de transfert.

1. Évaluer le temps de l'opération effectuée au niveau du serveur.
2. Évaluer le temps de l'opération effectuée au niveau du client.
3. Effectuer la moyenne des deux précédents relevés.

En effet l'opération au niveau du serveur consiste à fragmenter le fichier demandé en paquets puis d'envoyer au fur et à mesure. Son temps d'exécution est dont strictement inférieur au temps total de l'opération. Au niveau du client, il s'agit de recevoir ses paquets, et de les assembler au fur et à mesure. Son temps d'exécution majore dont le temps de l'opération. Or c'est bien les temps de transferts qui nous intéresse dans cette étude.

3.1.2 Fichiers supports

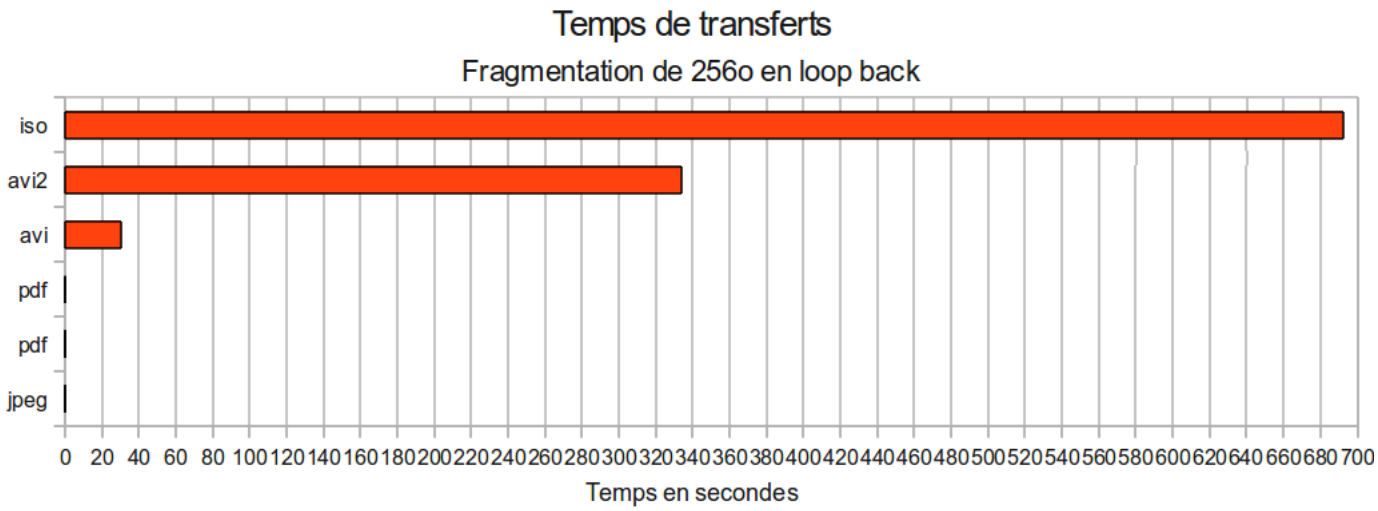
Pour ces jeux d'essais nous allons utiliser plusieurs fichiers dont les caractéristiques sont résumées ci-après :

Pour plus de clarté dans l'explication des tests, seuls les noms des fichiers apparaitront par la suite. Les différents tests tenteront d'illustrer l'impact des facteurs comme la taille des fichiers,

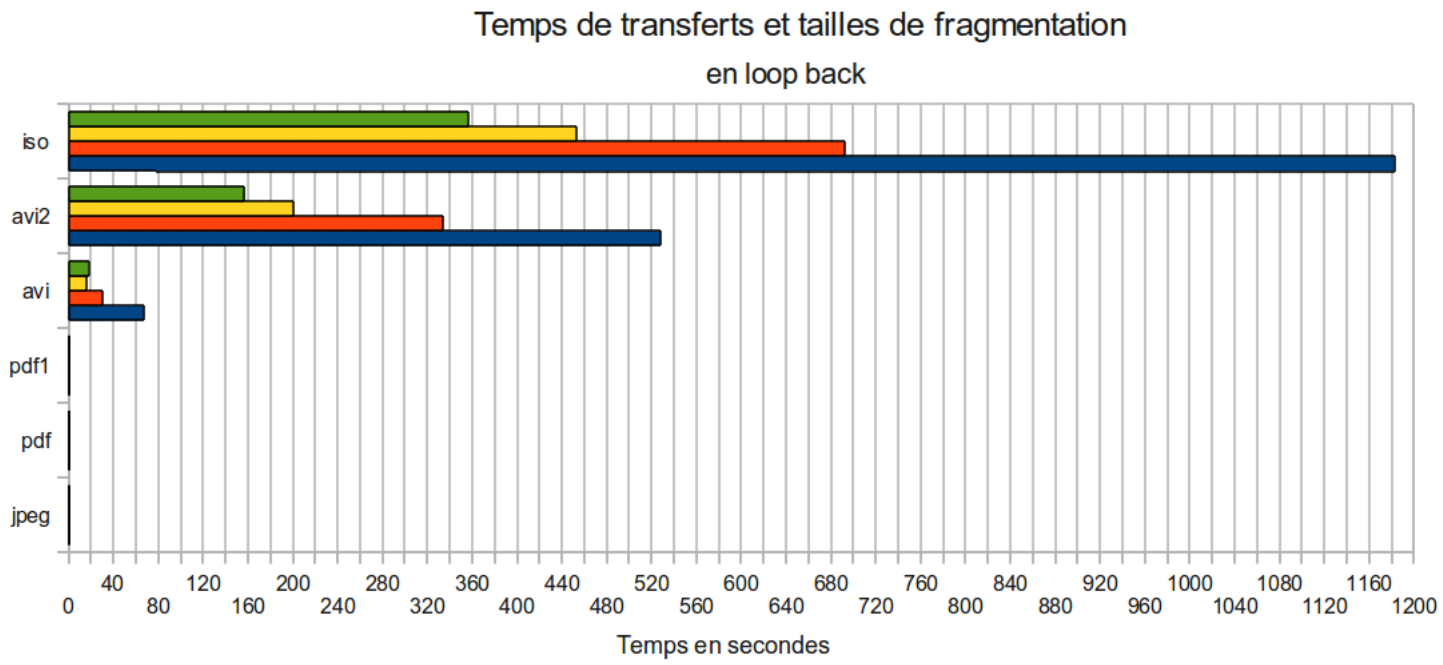
Caractéristiques / Fichiers	taille en Ko	extension
jpeg	6,090	jpeg
pdf1	158,410	pdf
pdf2	174,870	pdf
avi1	244482,316	avi
avi2	1942154,240	avi
iso	4345626,624	iso

celle de la fragmentation, ainsi que la distance physique entre le client et le serveur.

3.2 Influence de la taille des données à transférer



3.3 Variation de la taille des paquets



On constate que lors d'un transfert sur une même machine. L'augmentation de la taille de fragmentation permet une accélération de l'opération de transfert. Cependant pour une division supérieur à 1200 octets, des relevés de temps supplémentaires montre une stagnation des temps. Cela peut s'expliquer par le fait que le débit devient trop important pour les autres opérations (lectures/écriture dans le fichier ...). De la sorte, le temps minimum d'exécution est dépendant de ces opérations.

3.4 Autres tests

blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah

4 Guide d'utilisation

4.1 Manuel

4.2 Erreurs non prises en compte

Une grande partie des erreurs ignorées volontairement sont celles engendrées par le contrôles des informations saisies par l'utilisateur. La gestion de ce type d'erreurs se fait simplement avec des comparaisons de chaînes de caractères. Le but de ce projet étant avant tout la programation réseau nous avons décidé de ne pas perdre de temps sur ce type opérations. Nous avons eu l'occasion à de très nombreuses reprises de le faire dans la rédaction de notre code.

- Pour les opération de download et upload. Il est impératif de d'entrer le nom exact du fichier.
- Dowload file : Une fois sélectionée, l'utlisateur devra taper le nom exact du fichier à télécharger. Eventuellement il peut copier coller le nom du fichier qui l'intéresse à partir du résultat du listing du serveur qu'il aura opéreé au préalable.
- Upload file : Une fois sélectionée, l'utlisateur devra taper le nom exact du fichier qu'il veut transmettre au serveur. Celui sera stocjé dans le dossier du serveur selon la syntaxe suivante *outnomdefichier*.
- Enter a commande : Permet d'entrer une commande sans paramètres : ls, du pwd ...
- Quit : Le client se déconnecte du serveur et se termine.
- Eteindre le server : Permet de terminer le processus serveur. Bien que non cohérent avec le principe d'utilisation d'un serveur FTP, cela est pratique pour des tests. Cela permet entre autres de fermer correctement les différentes socket pour relancer de nouveaux serveurs.