Université Grenoble Alpes
Licence d'Informatique L3

Introduction aux Systèmes et Réseaux

TP : Transfert de fichiers (*FTP)*

L'objectif du TP est de construire un serveur de fichiers inspiré par le protocole FTP [1]. Les différentes sections correspondent aux différentes étapes du projet.

Traiter les étapes dans l'ordre de leur présentation.

Rendu de projet Les inscructions détaillées sont sur Moodle. Chaque binôme devra rendre le code source commenté des programmes réalisés et un un compte-rendu présentant : (a) les principales réalisations, en explicitant l'approche, l'architecture, l'interconnexion des entités et (b) une description des tests effectués. Une démonstration des réalisations sera organisée en fin de semestre (détails indiqués en temps utile).

Date limite 2 avril 2022 au soir.

1. `https://www.ietf.org/rfc/rfc959.txt`

1 Etape I : Serveur FTP de base

La première étape consiste à fournir une version de base d'un serveur de fichier. Dans cette version (1) le client demande un fichier au serveur en spécifiant le nom du fichier et (2) le serveur renvoie son contenu si le fichier est disponible.

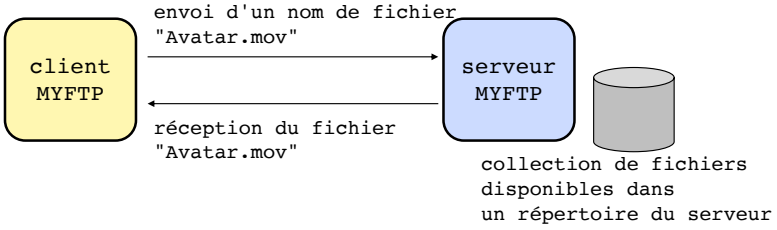

FIGURE 1: Schéma de base client-serveur de transfert de fichiers

Question 1 Mise en place d'un serveur FTP concurrent simple

Fournir une première implémentation de l'échange client-serveur pour le transfert de fichiers. Pour ce faire, récupérer la version concurrente avec pool de processus que vous avez mis en place pour le serveur ECHO (cf. TP 7) et changer l'interaction entre le client et le serveur comme suit :

1. **Port d'écoute du serveur 2121.** Le serveur de fichiers va écouter sur le port prédéfini 2121. Vous n'avez donc pas besoin de le passer en paramètre lors du lancement du serveur et des clients.
2. **Pool de processus parametré à l'aide de NB_PROC.** Le nombre de processus dans votre pool de processus doit être configuré à l'aide d'une constante `NB_PROC` et non écrit en dur dans le code.
3. **Terminaison propre du serveur** Le serveur doit pouvoir être arrêté proprement. cet arrêt doit se charger du nettoyage de tous les processus exécutants du pool de processus. Pour se faire, le serveur retransmettra le signal `SIGINT` à chacun de ses fils via la programmation d'un traitant de signal approprié (pas de zombis, cf. TP 7).
4. **Un fichier par connexion.** Quand le client se connecte au serveur, il ne lui demande qu'un seul fichier. La connexion entre un client et le serveur est donc terminée après un unique échange.
5. **Traitement de la requête côté serveur.** Après réception du nom de fichier demandé par le client, le serveur le charge en mémoire **en une seule fois**. Il le transmet ensuite dans le flot TCP à destination de celui-ci. Si le fichier n'existe pas, le serveur renvoie un code erreur.
6. **Types de fichiers considérés.** Le client doit pouvoir récupérer tout type de fichier, y compris les fichiers binaires (images, films, PDF, ...).

7. **Dossiers de stockage des fichiers.** Le serveur sert les fichiers depuis un répertoire local. Le client sauvegarde les fichiers reçus dans un répertoire local. Si vous testez votre projet sur une seule machine, bien faire attention à ce que les deux répertoires soient différents.
8. **Fin de traitement de requête côté client.** Un message sur la sortie standard du client confirmera la bonne réception du fichier. Ce message contiendra également des statistiques sur l'envoi (voir exemple ci-dessous). En cas d'erreur (e.g., fichier manquant, crash du serveur), le client affichera un message d'erreur et terminera proprement son exécution.

Exemple d'une exécution correcte :

```
$ ./clientFTP mon_serveur_FTP
Connected to mon_serveur_FTP.
ftp> get Nom_du_fichier
Transfer successfully complete.
X bytes received in Y seconds (Z Kbytes/s).
```

2 Etape II : Amélioration du serveur FTP

Question 2 Découpage du fichier.

Modifier le chargement du fichier en mémoire pour le charger par blocs de taille fixe (au lieu de le charger en une seule fois). L'envoi du fichier au client se fera donc par blocs. L'objectif étant de ne pas monopoliser la mémoire lors du transfert de fichiers volumineux.

Question 3 Plusieurs demandes de fichiers par connexion.

Modifier votre implémentation pour permettre aux clients d'effectuer plusieurs demandes de fichiers les unes après les autres, sans renouveler la connexion avec le serveur FTP. Plus précisément, le serveur FTP ne devra plus fermer la connexion après avoir envoyé le fichier au client. A la place, la connexion sera terminée par le client en utilisant la commande `bye`.

Question 4 Gestion des pannes coté client.

Modifier la version précédente en y ajoutant une gestion des pannes coté client. Si un client crashe pendant un transfert, le serveur doit continuer à fonctionner correctement en nettoyant proprement les structures système. Si le client est relancé, il doit reprendre le transfert qui a été interrompu (ne télécharger que la partie manquante du fichier).

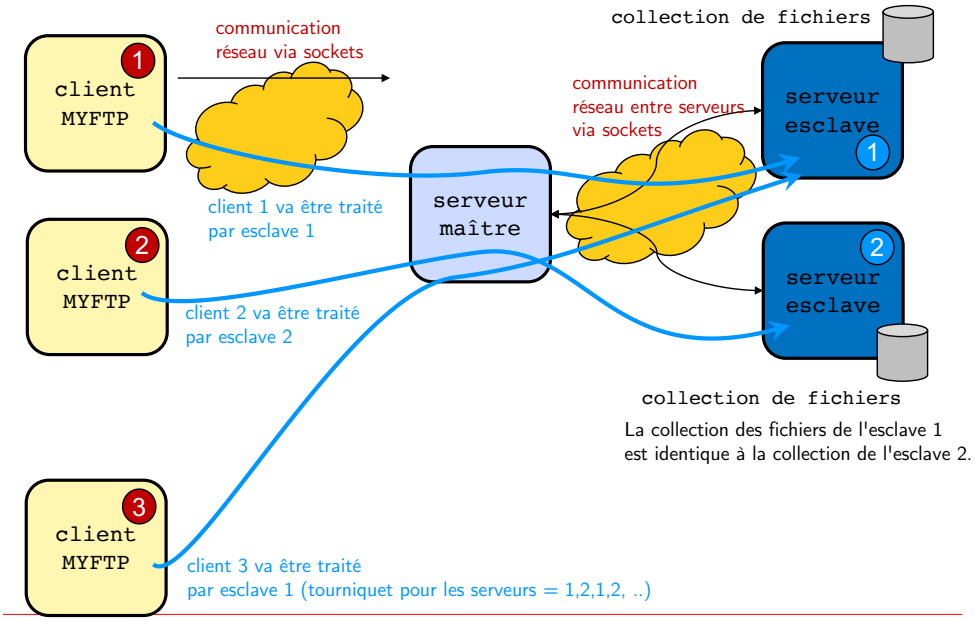

FIGURE 2: Schéma représentant l'architceture FTP avec répartiteur de charge. Dans cet exemple il y a deux serveurs esclaves et trois clients. Le premier client va interagir avec le premier esclave, le deuxième avec le deuxième esclave, le troisième de nouveau avec le premier et ainsi de suite (tourniquet). Les interactions entre clients et serveurs esclave ne passent pas forcément par le serveur maître.

3 Etape III : Répartition de charge entre serveurs FTP

Dans cette question nous voulons gérer un ensemble de serveurs qui peuvent gérer plusieurs clients à la fois. Ainsi, quand un client se connecte, il se connecte à un premier serveur maître qui joue le rôle d'un répartiteur de charge (*load balancer*). Ce serveur principal connaît un ensemble de serveurs esclaves qui sont capables de traiter les requétes des clients. Quand un client se connecte, le maître choisit un esclave qui va s'occuper de ce client. Le maître répartit les clients entre les esclaves en suivant une politique *Round-Robin* (un tourniquet simple). On considère que tous les serveurs esclaves disposent de tous les fichiers (les fichiers sont dupliqués). Le maître et les esclaves communiquent via des sockets.

Question 5 Connexion entre serveur maître et serveurs esclaves. Pour construire le système demandé, proposer une solution qui permet de définir l'ensemble de serveurs esclaves. Comment le serveur maître va les connaître ? Comment faudrait-il lancer le système (tous les serveurs) ? Comment le serveur maître va-t-il être connecté aux serveurs esclaves ?

Question 6 Protocole d'interaction entre client et serveur esclave Proposer un protocole pour l'établissement de la connexion entre un client et le serveur esclave qui est choisi par le maître. En d'autre termes, une fois que le client est connecté au maître et que le maître choisit le serveur esclave, comment le client fera pour interagir avec l'esclave ?

4 Etape IV : Opérations avancées

ATTENTION : ne pas traiter cette étape si l'étape précédente n'est pas faite.

Question 7 Commandes `ls`, `pwd` et `cd`

Modifier la version précédente afin d'enrichir votre serveur FTP avec les commandes suivantes :

— `ls` : retourne le contenu du répertoire courant du serveur FTP,
— `pwd` : affiche le chemin courant du serveur FTP,
— `cd` : change le dossier courant sur le serveur FTP.

Le serveur maître et ses esclaves ont donc une arborescence correspondant à une partie du système de fichier.

Question 8 Commandes `mkdir`, `rm`, `rm -r` et `put`

Enrichir le serveur FTP avec les commandes suivantes :

— `mkdir` : permet de créer un dossier sur le serveur FTP,
— `rm` : permet de supprimer un fichier sur le serveur FTP,
— `rm -r` : permet de supprimer un dossier sur le serveur FTP,
— `put` : permet de téléverser un fichier sur le serveur FTP.

Si un client effectue ce genre de requêtes, le serveur auquel le client est connecté doit répercuter ces modifications sur tous les autres serveurs esclaves (chaque système de fichier doit être mis à jour au bout d'un temps fini). On ne demande pas de cohérence forte ici (si un client se reconnecte, il n'a pas la garantie de voir ces modifications immédiatement prises en compte).

Question 9 Authentification

Les commandes implémentées dans l'étape précédente modifient les fichiers auxquels un serveur FTP donne accès et sont donc dangereuses. Pour protéger le serveur contre leur utilisation malveillante, mettre en place un système d'authentification où seulement certains utilisateurs (login :mot de passe) pourront les utiliser. Une tentative d'utilisation de ces commandes sans authentification préalable devrait échouer.