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.
- Le compte-rendu doit présenter :
  a. les principales réalisations, en explicitant l'approche, l'architecture, l'interconnexion des entités et
  b. une description des tests effectués.

Date limite

- 29 mars 2025 pour G1 et G2
- 31 mars 2025 pour G3 et G4

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.

Question 1 Définir un type énuméré `typereq_t` pour encoder les différents types de requêtes qu'un client peut envoyer vers le serveur. En fonction de votre avancement, ce type définira des valeurs comme `GET`, `PUT` ou `LS`. Dans cette première phase, vous n'avez à traiter que la requête de type `GET`.

Question 2 Définir un type de structure de données `request_t` pour formatter les requêtes client vers le serveur. Cette structure doit avoir au moins deux champs :

- un champ entier qui correspond au type de la requête.
- un champ qui correspond à un nom de fichier.

Question 3 Squelette de code adapté. Reprendre le code de l'exemple client- serveur `echo` et définir le squelette du code pour le client-serveur FTP.

- **Renommage de fichiers.** Vous devez veiller à renommer vos fichiers en conséquence. Typiquement, le fichier de votre serveur ne doit pas s'appeler `echoserveri.c` puisqu'il ne concerne pas `echo` et n'est pas itératif.
- **Pool de processus.** Vous devez avoir une version de serveur concurrent avec un *pool* de processus. 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.
- **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.

Question 4 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.

Question 5 Dossiers de travail pour le client et le serveur Le serveur sert les fichiers depuis un répertoire local à la machine sur laquelle il s'exécute. Le client, de son côté, sauvegarde les fichiers reçus dans un répertoire local sur la machine sur laquelle il s'exécute. Définir deux répertoires par défaut à utiliser par le client et le serveur de manière à ce que cela permette une exécution dans le cas où le client et le serveur s'exéutent sur la même machine.

Question 6 Traitement côté serveur

- **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.
- **Réception d'une requête.** Après réception du la requete du client, le serveur doit identifier le type de requête et effectuer le traitement en conséquence. Si le type de requête est invalide, le serveur doit renvoyer une réponse indiquant une erreur au

client. Voir la question suivante.

— **Réponse à une requété.** Définir un type de structure de données `response_t` pour les réponses du serveur au client. Cette structure doit avoir au moins un champ entier qui correspond à un code de retour indiquant succès ou erreur.

— **Traitement de la requête `GET`.** Dans le cas d'une requête `GET`, le serveur doit charger le fichier demandé en mémoire **en une seule fois**. Il le transmetta ensuite au client dans le flot TCP à destination de celui-ci. Si le fichier n'existe pas, le serveur doit renvoyer un code erreur.

— **Types de fichiers considérés.** Le client doit pouvoir récupérer tout type de fichier, y compris les fichiers binaires (images, films, PDF, ...).

Question 7 Traitement côté client

— **Encodage de la requête.** En fonction de la saisie de l'utilisateur, le client doit préparer la requête sous forme de structure `request_t` et l'envoyer au serveur.

— **Réception de réponse.** Un message sur la sortie standard du client indiquera le résultat de l'échange. En cas d'erreur, le client doit afficher un message d'erreur pertinent. En cas de succès, un message doit confirmer la bonne réception du fichier. Ce message contiendra également des statistiques sur l'envoi (voir exemple ci-dessous). Dans tous les cas, le client doit terminer 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 8 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. Vous devez définir le protocole que suivrons le client et le serveur dans ce cas là : (1) quelle est la taille du bloc ? (2) comment le client saura combien de blocs devra-t-il recevoir ?

Question 9 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 10 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,

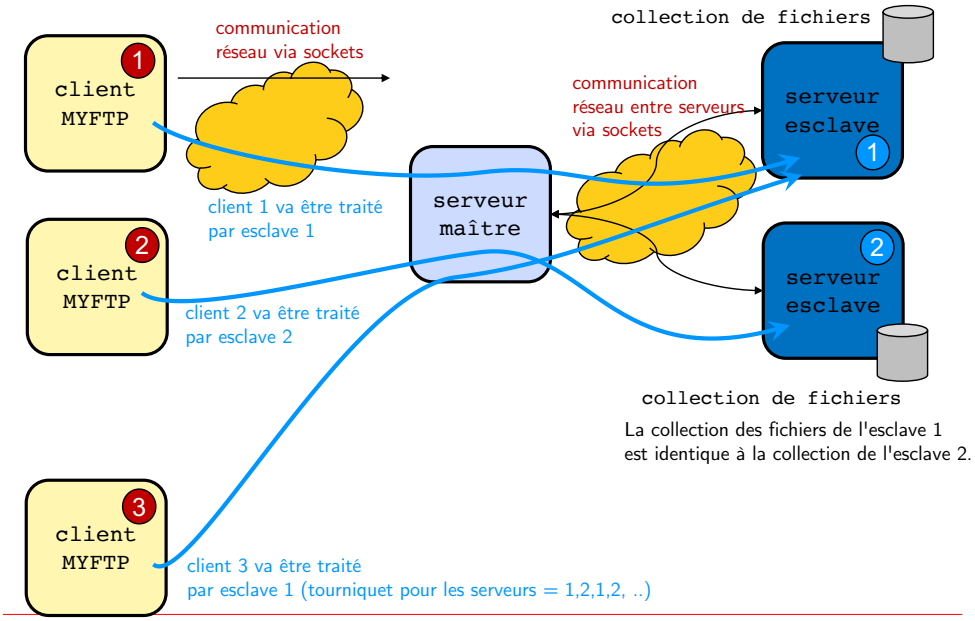

FIGURE 1: Schéma représentant l'architceture FTP avec répartiteur de charge. Dans cet exemple il y a deux serveurs esclaves et trois clients. Le premier client va interagir avec le premier esclave, le deuxième avec le deuxième esclave, le troisième de nouveau avec le premier et ainsi de suite (tourniquet). Les interactions entre clients et serveurs esclave ne passent pas forcément par le serveur maître.

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). Pour cela, le client doit garder des informations en local sur l'avancement du transfert.

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 requêtes 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).

Question 11 Au niveau du serveur maître, définir un nombre statique d'esclaves `NB_SLAVES` et choisir un port de connexion spécifique et surtout différent de `2121` pour les serveurs esclaves.

Question 12 Interconnexion entre maître et esclaves. Le serveur mâitre doit établir `NB_SLAVES` connexions avec `NB_SLAVES` serveurs esclaves avant de pouvoir répondre

à des demandes de connexions de clients. Pour chaque serveur eclave, il doit récupérer et garder les informations pertinentes (lesquelles ?) qui seront nécessaires à la connexion d'un client avec ce serveur esclave.

Question 13 Adaptation de la phase de connexion d'un client. Le protocole initial d'échange entre client et serveur doit être adapté. En effet, après connexion d'un client au serveur maître, il doit recevoir les informations de connexion pour pouvoir se connecter à un serveur esclave. Après réception de ces informations, il se connecté à un serveur esclave, et suivra le protocole d'échange des versions précédentes.

Question 14 Bonus : reconnexion en cas de panne de serveur esclave. Proposer une solution (un protocole d'échanges) afin de mettre en place une redirection de client vers un serveur esclave fonctionnel en cas de panne du serveur esclave auquel il est connecté.

4 Etape IV : Opérations avancées

ATTENTION : ne pas traiter cette étape si l'étape précédente n'est pas faite. Si cette consigne n'est pas respectée, cela se traduira par un malus.

Question 15 Commandes `ls`

Modifier la version précédente afin d'enrichir votre serveur FTP avec la commande `ls` qui retourne le contenu du répertoire courant du serveur FTP. Vous pouvez implémenter cette partie en utilisant les principes de création de fils (`fork`), exécution de programme (`exec`) et redirection d'E/S (`dup2`). Vous pouvez également regarder et utiliser la fonction `popen`.

Question 16 Commandes `rm` et `put`

Enrichir le serveur FTP avec les commandes suivantes :

— `rm` : permet de supprimer un fichier 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 17 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.