Binome : Djaghloul Ayoub Ahmed Amine

Mekaoussi Mohamaed Fares

Rapport - TP Transfert de fichiers (FTP)

I. Première étape : Serveur FTP de base

Introduction :

Le but de ce TP est de mettre en place un serveur de fichiers inspiré du protocole FTP. Dans cette première étape, nous avons implémenté une version de base d'un serveur de fichiers qui permet au client de demander un fichier en spécifiant son nom et au serveur de renvoyer son contenu si le fichier est disponible.

Resumé de l'étape 1 :

- Il s'agit d'une implémentation d'un serveur FTP (File Transfer Protocol) simplifié. Le serveur peut envoyer des fichiers à un client qui se connecte au serveur via une connexion TCP.
- Le serveur utilise un processus fils pour gérer chaque client qui se connecte, et peut traiter plusieurs connexions simultanément grâce à un pool de processus.
- Le client peut demander un fichier en tapant le nom du fichier dans la console du client. Le serveur recherche le fichier dans le répertoire "Server/" et le renvoie au client. Si le fichier n'existe pas, le serveur envoie un message d'erreur au client. Si le client tape "bye", la connexion est fermée.
- Il y a des commentaires dans le code qui expliquent chaque partie en détail.

Organisation de notre système :

Notre système se compose de deux entités, à savoir le client et le serveur. Le serveur est l'entité principale qui gère les connexions avec les clients et la transmission des fichiers. Le client, quant à lui, est l'entité qui demande les fichiers au serveur et reçoit les fichiers.

Connexions entre les entités :

- Le client se connecte au serveur en appelant la fonction Open_clientfd() qui établit une connexion TCP/IP avec le serveur. Le serveur accepte ensuite cette connexion en appelant la fonction Accept().
- Une fois la connexion établie, le client envoie des commandes au serveur via la fonction Rio_writen() qui écrit des données sur la socket, et le serveur reçoit ces commandes via la fonction Rio_readlineb() qui lit des données à partir de la socket.
- Le serveur répond ensuite aux commandes du client en envoyant des données via la fonction Rio_writen() et le client les reçoit via la fonction Rio_readnb().
- Enfin, une fois la transaction terminée, le client et le serveur ferment la connexion en appelant la fonction Close().

Déroulement du système :

Le déroulement du système serait le suivant :

1. Le serveur est exécuté et écoute sur un port donné (par exemple 2121).

2. Le client est exécuté et se connecte au serveur en utilisant le port et l'adresse IP du serveur.
3. Le client envoie une commande au serveur pour demander l'envoi d'un fichier, en utilisant le protocole FTP et en utilisant la commande "nom_du_fichier"
4. Le serveur reçoit la commande, vérifie si le fichier existe et s'il existe, envoie le contenu du fichier au client.
5. Le client reçoit le contenu du fichier et le stocke dans un nouveau fichier dans le répertoire "Client/" avec le même nom que le fichier original.
6. Le client affiche un message de confirmation indiquant que le fichier a été téléchargé avec succès.
7. La connexion entre le client et le serveur est fermée.
8. Le client se termine.
9. Le serveur continue à écouter sur le port donné pour de futures connexions.

Rapport Étape II : Amélioration du serveur FTP

Dans cette étape, nous avons apporté plusieurs améliorations à notre serveur FTP pour le rendre plus robuste et capable de gérer des transferts de fichiers volumineux. Nous avons implémenté trois fonctionnalités principales :

1. **Découpage du fichier** : nous avons modifié le chargement du fichier en mémoire pour le charger par blocs de taille fixe, au lieu de le charger en une seule fois. Cela permet de ne pas monopoliser la mémoire lors du transfert de fichiers volumineux. L'envoi du fichier au client se fait donc par blocs.
    - **Coté serveur :**

      Dans le code, la taille des blocs est définie par la constante BLOCKSIZE. La fonction utilise une boucle pour lire le fichier en blocs de taille fixe et les envoyer au client. La boucle continue jusqu'à ce que tous les blocs aient été lus et envoyés ou qu'une erreur de lecture se produise.
    - **Coté client :**

      La taille des blocs est définie par la constante BLOCKSIZE, Le client utilise une boucle pour lire les blocs de données envoyés par le serveur en blocs de taille fixe et les écrire dans un fichier sur le client. La boucle continue jusqu'à ce que tous les blocs aient été lus et écrits.

2. **Multiples demandes de fichiers par connexion** : nous avons modifié notre implémentation pour permettre aux clients d'effectuer plusieurs demandes de fichiers les unes après les autres, sans avoir à renouveler la connexion avec le serveur FTP. Ainsi, le serveur FTP ne ferme plus la connexion après avoir envoyé le fichier au client. La connexion est terminée par le client en utilisant la commande "bye".
    - **Coté client :**

Nous devons ajouter une boucle while dans le code client pour permettre à l'utilisateur de saisir plusieurs demandes de fichiers l'une après l'autre, Nous utilisons ensuite une boucle while pour lire les données du fichier par blocs de BLOCKSIZE octets. À chaque itération de la boucle, nous calculons le nombre d'octets restants à lire et lisons jusqu'à BLOCKSIZE octets. Nous suivons le nombre total d'octets lus jusqu'à présent en utilisant la variable total_read.

- **Coté serveur :**

Nous avons ajouté une boucle while qui lit les commandes du client à partir de la connexion. Si la commande est "bye\n", la boucle se termine et la fonction ftp() se termine, mais la connexion reste ouverte. Si la commande est un nom de fichier, la fonction ftp() envoie le contenu du fichier au client, puis attend la commande suivante. Si le fichier n'existe pas, la fonction envoie la chaîne "ERR\n" au client.

3. **Gestion des pannes côté client** :

- **Coté Client**

Avant de demander le fichier au serveur, vérifiez si le fichier existe déjà localement. Si oui, récupérez la taille du fichier et envoyez cette information au serveur.
Lors de la réception du fichier, reprenez le transfert en fonction de la taille du fichier déjà téléchargée à l'aide du seek pour decaler vers la bonne position dans le fichier.

- **Coté Serveur**

Lisez la taille du fichier déjà téléchargée envoyée par le client.
Lors de l'envoi du fichier, commencez à envoyer les données à partir de la position indiquée par le client à l'aide du seek pour decaler vers la bonne position dans le fichier.

<u>Rapport d'étape 3 : Répartition de charge entre serveurs FTP</u>

Dans le cadre de notre projet, nous avons étudié la répartition de charge entre serveurs FTP. Plus précisément, nous avons exploré comment gérer un ensemble de serveurs pour traiter les demandes de plusieurs clients en utilisant une politique de répartition de charge RoundRobin.

Dans cette étape, nous avons appris comment gérer un ensemble de serveurs FTP en utilisant un serveur principal comme répartiteur de charge. Les serveurs esclaves sont capables de traiter les requêtes des clients et le serveur maître choisit un esclave qui va s'occuper de chaque client. Tous les serveurs esclaves disposent de tous les fichiers, qui sont dupliqués.

Nous avons compris le fonctionnement de cette méthode de répartition de charge, mais n'avons pas eu l'opportunité de l'implémenter dans notre projet.

Question 5 : Connexion entre serveur maître et serveurs esclaves

Pour construire le système demandé, il faudrait définir l'ensemble de serveurs esclaves. Une solution consiste à spécifier une adresse IP et un numéro de port pour chaque serveur esclave. Le serveur maître pourrait les connaître en les stockant dans une base de données ou un fichier de configuration.

Pour lancer le système, tous les serveurs doivent être démarrés simultanément. Le serveur maître peut ensuite se connecter à chaque serveur esclave en utilisant leur adresse IP et leur numéro de port.

Question 6 : Protocole d'interaction entre client et serveur esclave

Une fois que le client est connecté au maître et que le maître a choisi le serveur esclave, le client doit pouvoir interagir avec cet esclave. Pour cela, nous proposons un protocole d'interaction basé sur le protocole FTP standard.

Lorsque le maître choisit un esclave pour traiter la demande du client, il envoie un message au client contenant l'adresse IP et le numéro de port de l'esclave. Le client se connecte ensuite à l'esclave en utilisant cette information.

Une fois la connexion établie, le client peut envoyer des commandes FTP standard à l'esclave pour télécharger, téléverser ou supprimer des fichiers. L'esclave traite ensuite les commandes et envoie les réponses correspondantes au client.

En conclusion, la répartition de charge entre serveurs FTP est une méthode efficace pour traiter les demandes de plusieurs clients en utilisant plusieurs serveurs. Bien que nous n'ayons pas pu implémenter cette méthode dans notre projet, nous avons appris comment définir l'ensemble de serveurs esclaves et comment établir un protocole d'interaction entre le client et l'esclave choisi par le maître. Ces connaissances pourraient être utiles dans le développement de systèmes de serveurs FTP à grande échelle.

Problèmes rencontrés :

Etape 2 :

Dans l'étape 2 de notre projet, nous avons rencontré un problème lié à la gestion des erreurs lorsqu'un fichier n'existe pas. En effet, si nous testons l'existence d'un fichier qui n'existe pas, le système retourne une erreur de type "pipe error".

Lors de la mise en place de la gestion des pannes côté client, nous avons rencontré un problème lié à la transmission de la taille du fichier déjà téléchargé au serveur. En effet, malgré nos efforts, nous n'avons pas réussi à mettre en place une méthode pour envoyer cette information au serveur.

Malheureusement, en raison de contraintes de temps, nous n'avons pas été en mesure d'implémenter ces fonctionnalités dans notre projet. Cependant, nous sommes conscients de l'importance de la gestion des pannes dans un contexte de développement de systèmes de serveurs FTP à grande échelle et nous continuerons à explorer des solutions pour améliorer la fiabilité de notre système dans les travaux futurs.