SYSTÈMES ET RÉSEAUX: SERVEUR FTP

RAPPORT DE PROJET

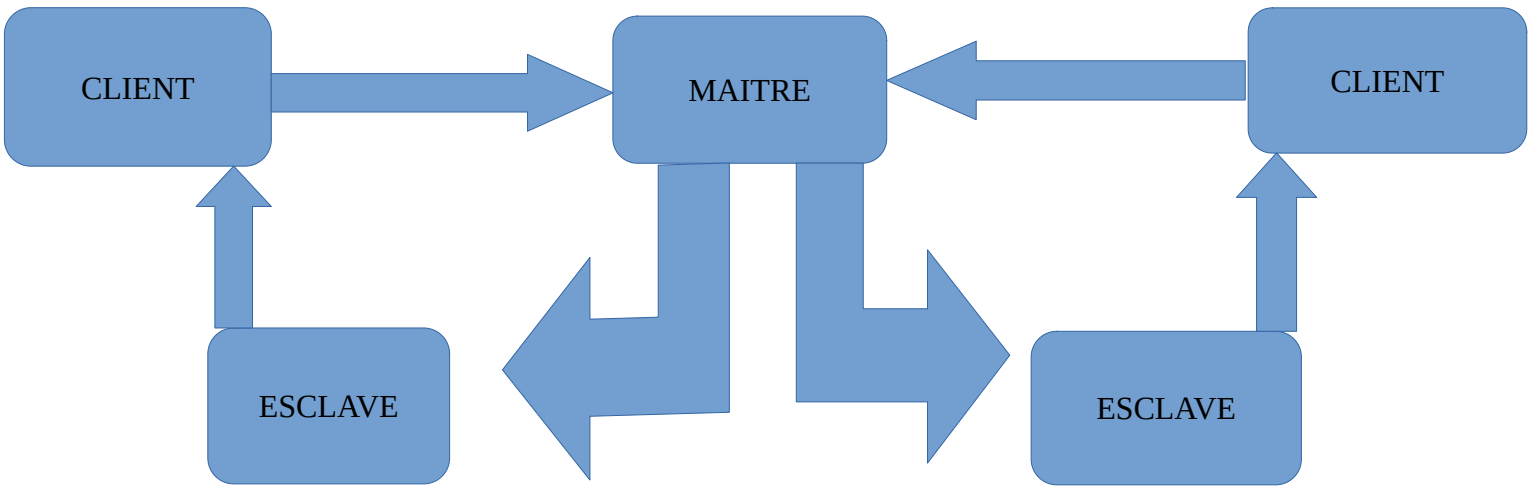
FAROUMA TALL _ MAMADOU LAMINE BAH

Introduction :

Ce projet consiste à implémenter un serveur similaire au serveur FTP.Ce dernier possédera une partie serveur multiprocessus comprenant un certain nombre d'optimisations.

1. **Principales réalisations**

- **Serveur FTP concurrent :**
  Pour cette fonctionnalité,le sujet du TP7 a été repris : nous avons mis en place un serveur multi-processus qui crée NPROC processus qui attendent la connexion de clients.lorsqu'un client se connecte,le processus exécutant attend un nom de fichier.Lors du téléchargement du fichier ,la progression s'affiche et montre les informations sur le transfert.
- **Découpage du fichier téléchargé :**
  afin d'optimiser la mémoire lors des téléchargements de grands fichiers,le transfert des données est effectué par paquets de 9 octets.Nous lisons donc le fichier par blocs et cela jusqu'à ce que la fin du fichier soit atteinte,évitant d'envoyer tout le fichier d'un coup.
- **Gestion simple des pannes coté client :**
  Lorsqu'il y' a un crash et qu'on relance ensuite le téléchargement,nous avions déjà recupéré une taille courante qui est le nombre d'octets déjà téléchargés et donc on compare cette taille avec celle du fichier demandé.Vu qu'elles seront différentes,le téléchargement continue à partir de cette taille afin de télécharger le reste .
- **Serveurs FTP avec équilibrage de charge :**
  Pour mettre en place notre répartiteur de charge,nous avons mis en place le schéma suivant :



Lorsqu'un client se connecte au serveur,ce dernier se connecte à un esclave disponible en lui envoyant les informations du client(son port et son adresse IP).Dès lors,il est redirigé vers ce serveur esclave(Politique de Round Robin simple) puis le serveur est de nouveau disponible attendant l'arrivée d'un probable autre client. L'esclave se connecte au client à son tour.Il y'a donc un canal de communication bidirectionnel entre le client et le serveur esclave.Le même principe se déroule pour chaque client qui arrive.Il faut noter juste que le nombre de serveurs esclaves doit être égal à celui de clients et que la connexion entre serveur maitre et esclave n'est faite que lorsqu'un client se connecte.
Dans le serveur,une socket est créée pour la communication avec le client.Le client envoie au serveur son port et son host qui servira pour la connexion du client avec le serveur esclave.Une socket est aussi créée dans le serveur esclave et le serveur maitre devient son client.Il lui envoie ensuite les informations du vrai client.Ce client crée une socket pour pouvoir communiquer avec le serveur esclave.

- **Plusieurs demandes de fichiers par connexion :**non fonctionnel

Nous avons mis nos fonctions d'attente de commandes (coté client) et d'attente de requêtes (coté serveur) dans des boucles infinies, qui ne sont interrompues qu'en cas de déconnexion du client.Mais nous rencontrons toujours un problème lorsque le client veut demander une 2^{e} requête et nous ne savons pas d'où vient cette erreur.

- **Commandes ls,pwd,cd ,mkdir,rm,put :**

Ces commandes ont été implémentés dans le code de echo.c avec des primitives en c comme par exemple remove pour rm,chdir pour cd .
La bibliothèque dirent.h a été utilisée.

- **Authentification :** non implémenté

Idée:pour la gestion de l'authentification, nous avions eu comme idée de faire un système de compte utilisateur.Ainsi au démarrage du serveur, un fichier contenant la base de données des comptes sera lu et chargé en mémoire.
Lorsqu'un utilisateur entre une commande « authentification » par exemple suivie de son login, il sera invité à entrer son mot de passe, puis ce mot de passe sera immédiatement hashé coté client pour ne pas transiter en clair sur le réseau.Celui-ci sera ensuite comparé au hash contenu dans la base de données du serveur, et l'authentification réussira uniquement si les hashs sont identiques.
Pour éviter certaines attaques, le serveur déconnectera un client qui aura fait plus de trois tentatives d'authentification échouées.

2. Tests effectués

Comment compiler et exécuter notre programme ?
Exemple:**connexion avec 1 seul client**

- **serveur** : **1 - make clean**
  2 - make
  3 - ./slave <N°Port slave autre que 2121>
  4 - ./maitre <N°Port slave>

- **client** : **1 - make clean**
  2 - make
  3 - ./client <hostServeur> <hostSClient> <N°Port client>

Exemple :connexion avec 2 clients.

- **serveur** : **1 - make clean**
  2 - make
  3 - ./slave <N°Port slave1 autre que 2121>
  4 - ./slave <N°Port slave2 autre que 2121 et autre que port slave1>
  5 - ./maitre <N°Port slave1> <N°Port slave2>

- **client** : **1 - make clean**
  2 - make
  3 - ./client <hostServeur> <hostSClient1> <N°Port client1 autre que 2121>
  4 - ./client <hostServeur> <hostSclient2> <N°Port client2 autre que 2121>

Les fichiers avec lesquels nous testons sont un fichier .c,un son,un exécutable,un texte,un fichier zippé,une image...Ce dossier est appelé **test** et se trouve dans le serveur.

- **Test de transfert**
  On affiche chaque bloc de données envoyé sur la sortie standard suivi d'un court temps d'attente avant l'affichage du bloc suivant.
- **Test de panne**
  On simule une panne en faisant une frappe de CTRLZ au client lors du transfert d'un fichier . Ainsi à la reconnexion le téléchargement démarrera là où il s'était arrêté.
- **Test Load_Balencer**
  On teste avec 2 clients et 2 serveurs.
- **Test de commandes**
  Si on tape la commande mkdir <un_dossier> puis un ls on remarque bien que le répertoire « un_dossier » a bien été créé.

Conclusion :

Ce serveur FTP nous a permis de mieux comprendre les mécanismes mis au point par les serveurs pour leur permettre d'être le plus réactif possible.Nous pouvons dans notre cas envisager que le programme soit exécuté par un grand nombre de clients en même temps, et la présence du répartiteur de charge évite la limitation des processus. Ainsi la majorité des améliorations ont été implémentées .Les notions et fonctionnements des entités client et serveur sont bien assimilés et ont permis de comprendre l'échange d'informations ainsi que les vérifications effectuées par un serveur FTP classique.