HEULS Amandine
TERESE Niels

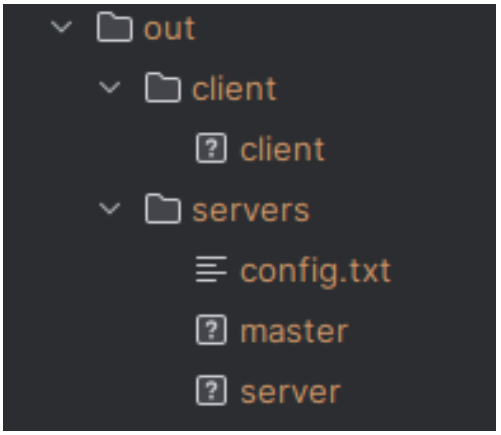
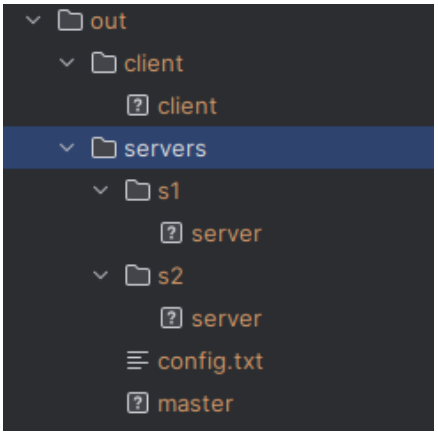
Compte-rendu TP FTP

Avancement du projet :

Nous avons traité toutes les questions jusqu'à la question 7 inclus.
Concernant la question 8, les commandes mkdir, rm / rm -r, ont été implémentées.
La commande “put” n’a pas été implémentée ainsi que la synchronisation des fichiers entre serveurs FTP.
Nous n’avons pas trouvé une bonne implémentation pour que chaque serveur FTP puisse “connaître” les autres existants et ainsi partager les modifications apportées dans le système de fichier par l’utilisateur.

Structure :

Pour compiler le projet, il faut se rendre dans “src”, puis faire un make.
Le makefile va ainsi créer la structure de gauche ci-dessous.

Structure du projet pour client / serveur. Ici le master n’a pas de vrai utilité car il y a un seul serveur ftp.	Voici un exemple de structure pour 2 serveurs FTP esclaves, 1 serveur Master, et 1 client. La création des dossiers s1 et s2 sont à faire manuellement ainsi que la copie de l'exécutable “server” dans ceux-ci.
	

Lorsque les exécutables “server” et “client” sont lancés, cela va créer un dossier “files” dans les répertoires courants respectifs.

Configuration et options :

Exécutable	Explications
./client	L'utilisateur doit obligatoirement fournir une adresse ip. Si aucun port n'est fourni, le port 2121 est celui par défaut (port du master) Si un port est fourni, on tente d'établir une connexion directe au serveur avec les paramètres données (ip et port). Exemple : ./client 127.0.0.1 ./client 127.0.0.1 2122
./server	L'utilisateur doit obligatoirement fournir un port. Le serveur FTP supporte 10 connexions simultanées via un pool de 10 processus. La taille de la pool de processus est configurable via la constante NB_PROC dans server.c Exemple : ./server 2122
./master	Le master se lance sans paramètre. Il utilise le port 2121 par défaut, modifiable via la constante PORT dans master.c Néanmoins, il va d'abord falloir remplir le fichier "config.txt" qui permet de lister les adresses ip et les ports respectifs des différents serveurs FTP. Le serveur Master supporte jusqu'à 5 serveurs "Slave". Cette limite est modifiable via la constante MAX_SLAVES disponible dans master.c Exemple : ./master

Fonctionnement :

Master :

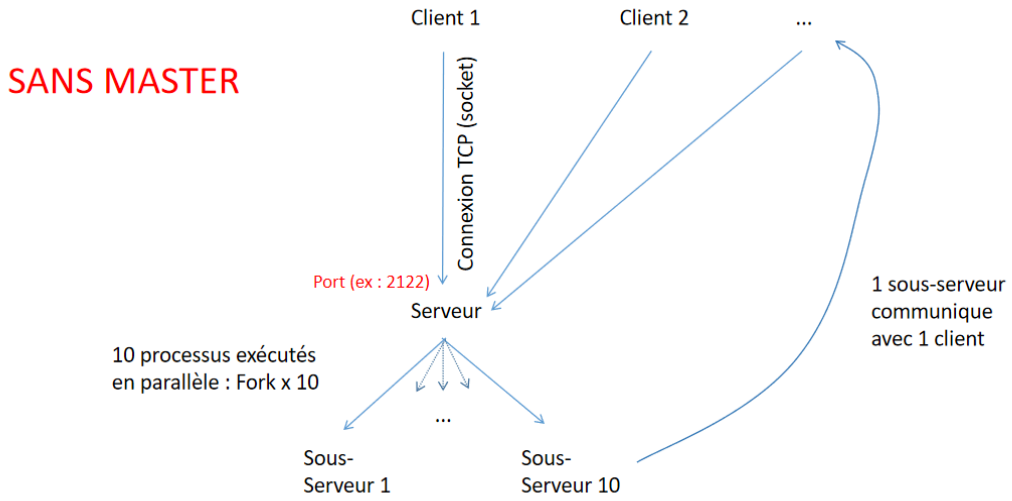
- Une structure interne ServerState permet de stocker de manière globale une structure Slave, comportant une adresse ip ainsi qu'un port, ainsi que le nombre de serveur esclave, et le numéro du prochain esclave afin de respecter une répartition de charge du type Round-Robin.
- Le rôle du Master est de rediriger le client vers un serveur FTP actif disponible. Pour cela, le master va recharger le fichier "config.txt" toutes les 10 secondes à l'aide d'un timer. Il parcourt ainsi les serveurs contenu dans "config.txt", détermine s'ils sont disponibles via un socket. Si c'est le cas, le master met à jour ServerState, et renvoie via la structure Slave une adresse ip et un port au client. Le client se déconnecte du master pour se connecter au serveur FTP slave fourni précédemment par le master.

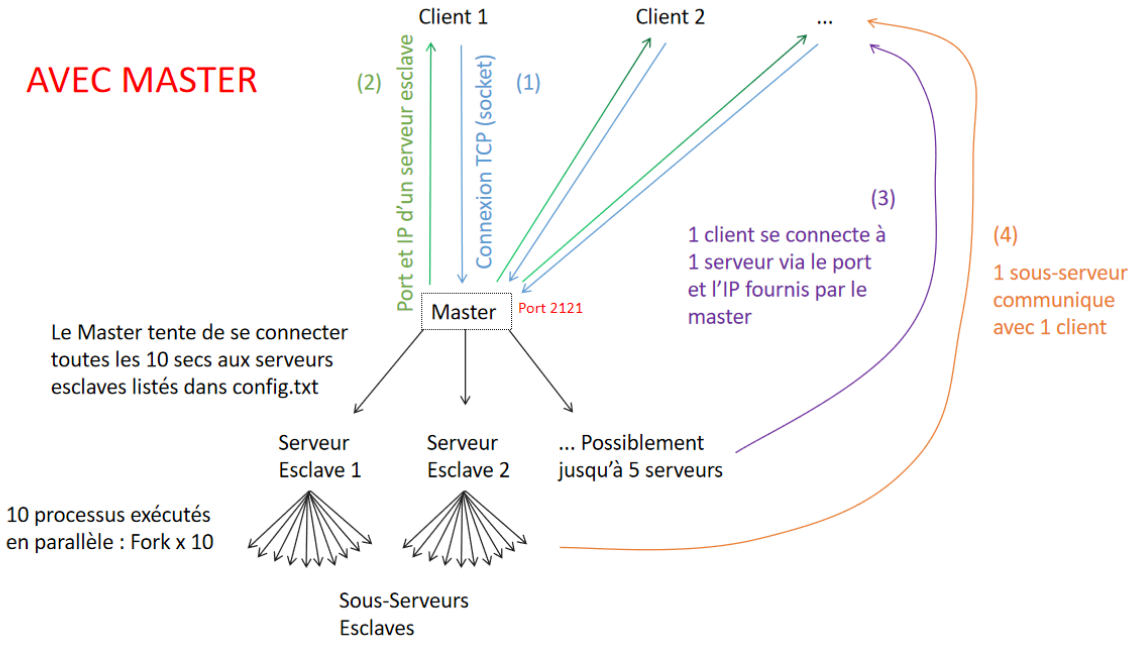
Client / Server :

- Une fois que le client est connecté au serveur FTP (voir le tableau de configuration), le client envoie une requête au serveur FTP via la structure Request dans protocol.c Elle comporte l'entrée de l'utilisateur, ainsi qu'un indice de bloc dans le cas d'une reprise de téléchargement d'un fichier.

- Le serveur récupère la requête de l'utilisateur, analyse et traite l'entrée du client. Il renvoie par le suite une structure Response au client comportant :
    - statut : 404 / 200 / 403 : non trouvé, trouvé, accès refusé
    - un message : le message qui vient avec le statut
    - le nombre de blocs : nombre de total blocs prévu dans le cas d'une requête d'un fichier
    - taille du fichier : la taille du fichier dans le cas d'une requête d'un fichier
- Dans le cas de l'envoi d'un fichier, après avoir envoyé la structure Response comportant les informations nécessaires, le serveur envoie le fichier demandé par bloc de taille BLOCK_SIZE (constante dans protocol.h) à l'aide d'une structure Block qui comporte :
    - buf : les données
    - size : la taille du buf de taille max BLOCK_SIZE
- Le client récupère la Response, affiche le message. Si la taille du fichier contenu dans la Response n'est pas égale à 0, on suppose alors qu'on a un fichier à récupérer. On lit donc les blocs afin de reconstituer le fichier, on enregistre dans un fichier temporaire en ".part" les blocs qu'on a réussi à lire et à écrire. Cela nous permet d'interrompre un téléchargement. Si on a pu récupérer l'intégralité des blocs, on supprime le fichier en ".part". A chaque lancement du client, on parcourt d'abord le dossier local "files" à la recherche de fichier ".part" afin de reprendre les éventuels téléchargements incomplets.

Les protocoles mis en place utilisent htonl et ntohl pour les champs qui ne sont pas du type char.





Sécurité :

Afin de sécuriser un minimum les serveurs FTP, la récupération des fichiers à l'aide de la commande : "get filename.extension" est passée à la fonction basename.
Celle-ci permet d'enlever les chemins relatifs et ainsi limiter la récupération au répertoire "files".
Exemple : /dir1/dir2/filename.extension => filename.extension

Concernant le changement de répertoire, on l'effectue avec chdir afin de changer le répertoire courant du processus courant. Ceci permet de lancer les prochaines commandes dans le nouveau répertoire.
Nous avons pris le soin de vérifier que l'utilisateur ne puisse pas se déplacer dans un dossier parent au répertoire "files" du serveur.