2022-2023

M1 BDGL UFR-MI UFHB

PROJET RESEAUX ET PROTOCOLES

AFRAITANE AYAL – TRAH YOUAN JUSTIN

# INTRODUCTION

Le but de cet exercice est de créer un serveur et établir la communication entre celui-ci et des clients afin de donner la possibilité aux clients d’avoir accès aux ressources stockées dans le serveur.

Dans notre travail notre serveur a été créé en python de même que les entités dites clientes au serveur.

Le serveur a des fichiers stockés dans un répertoire. Le client va pouvoir se connecter au serveur et exécuter une série de commandes soit pour :

* Obtenir de l’aide pour les commandes disponibles à envoyer au serveur.
* Obtenir la liste des fichiers stockés sur le serveur.
* Télécharger un ou plusieurs fichiers disponibles sur le serveur.
* Se déconnecter du serveur.

Les détails de leur création, leur mise en place, leur fonctionnement…etc. sont expliqués plus bas dans ce document.

# ARCHITECTURE

Comme précisé précédemment le travail a été fait en python. Il y a 2 script qui ont été créer. Un script pour le serveur et un pour le client. La communication utilise la bibliothèque ***socket*** de python pour lier les deux partis. Plus précisément on utilise ***socket.AF\_INET*** pour « Adress Family Internet » pour signifier que le type de socket utilisé est pour les connexions internet IPV4 et ***socket.SOCK\_STREAM*** pour signifier que le protocole de communication entre les deux partis est TCP qui est orienté connexion. De plus le serveur peut prendre en charge plusieurs clients simultanés grâce à la bibliothèque ***threading*** de python. A chaque nouvelle connexion le serveur associe un thread pour traiter le nouveau client connecté. On attribue une adresse IP et un port sur lequel il sera actif et en écoute pour toute communication. Coté client, pour pouvoir se connecter il lui suffit évidemment de connaitre l’adresse IP serveur et le port d’écoute du serveur pour se connecter.

# FONCTIONNEMENT

Notre serveur vient avec un dossier « server\_data » sur lequel sont stockés les fichiers qu’il contient. Ben sur on peut modifier le dossier qui contient les fichiers en modifiant la variable « SERVER\_DATA\_PATH » dans le script du serveur en faisant attention au chemin choisi. Ceci est valable pour le client qui vient avec un dossier « client\_data » ou seront stockés les fichiers qu’il aura téléchargé modifiable en modifiant la variable « CLIENT\_DATA\_PATH ».

Lorsqu’un client se connecte au serveur il faut bien sur établir les règles d’échanges des messages entre le serveur le client et la structure de ceux-ci.

C’est l’objet de notre protocole d’échange entre serveur et client.

Une fois la connexion TCP entre un client et le serveur établie,

Le client peut envoyer une série de commande dont la structure envoyée au serveur est :

**[Commande]@[option]**

Dans la pratique l’utilisateur n’aura qu’à taper (On ne tape pas les crochets):

**[Commande] [option]**

L’option n’est requise que pour télécharger un fichier. Elle correspond ici au nom du fichier à télécharger.

La liste des commandes disponibles est la suivante :

* LIST : retourne la liste des fichiers stockés par le serveur
* DOWNLOAD filename : télécharge un fichier du serveur
* LOGOUT : se déconnecter du serveur
* HELP : retourne la liste des commandes

La structure d’une réponse du serveur est la suivante à l’exception du téléchargement d’un fichier et de la déconnexion:

**[OK]@[réponse]**

Il n’est affiché à l’écran que la réponse du serveur.

Lors de la demande de téléchargement d’un fichier la structure de la réponse est :

**[Commande]@[filename]@[contenu]**

Et pour une déconnexion c’est le serveur qui reçoit juste la demande LOGOUT

Et c’est le client qui par la suite ferme la connexion.

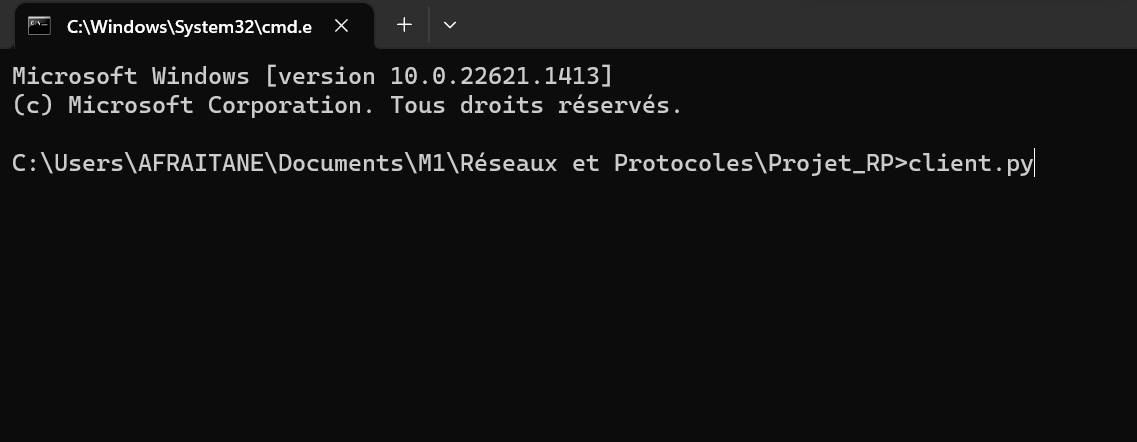
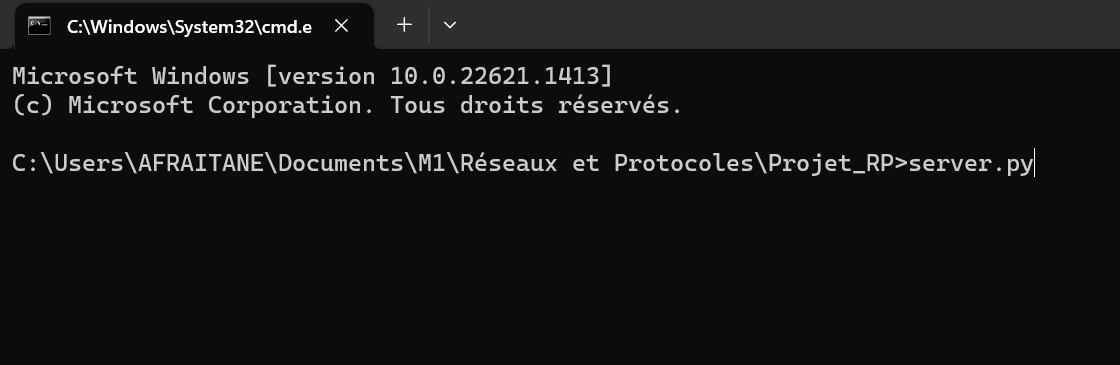
Si le client entre une commande qui n’est pas reconnue, le serveur lui retourne un message qui lui indique que ce n’est pas une commande valide et lui propose de taper HELP pour la liste des commandes.

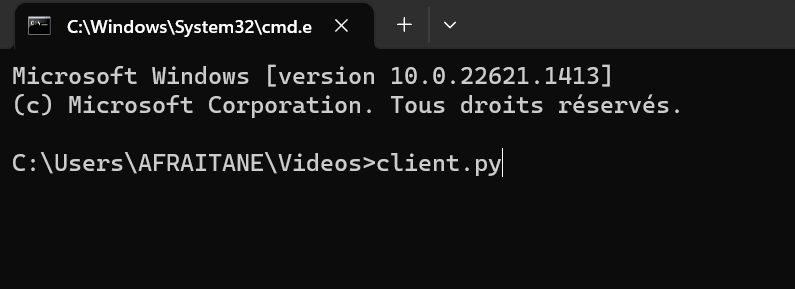
Comme le serveur gère plusieurs clients à la fois, chaque nouvelle connexion ou déconnexion est affichée coté serveur (adresse IP cliente et port client) ainsi que chaque requête d’un client avec son adresse et son port pour différencier la provenance des différentes requêtes.

# EXEMPLE D’UTILISATION

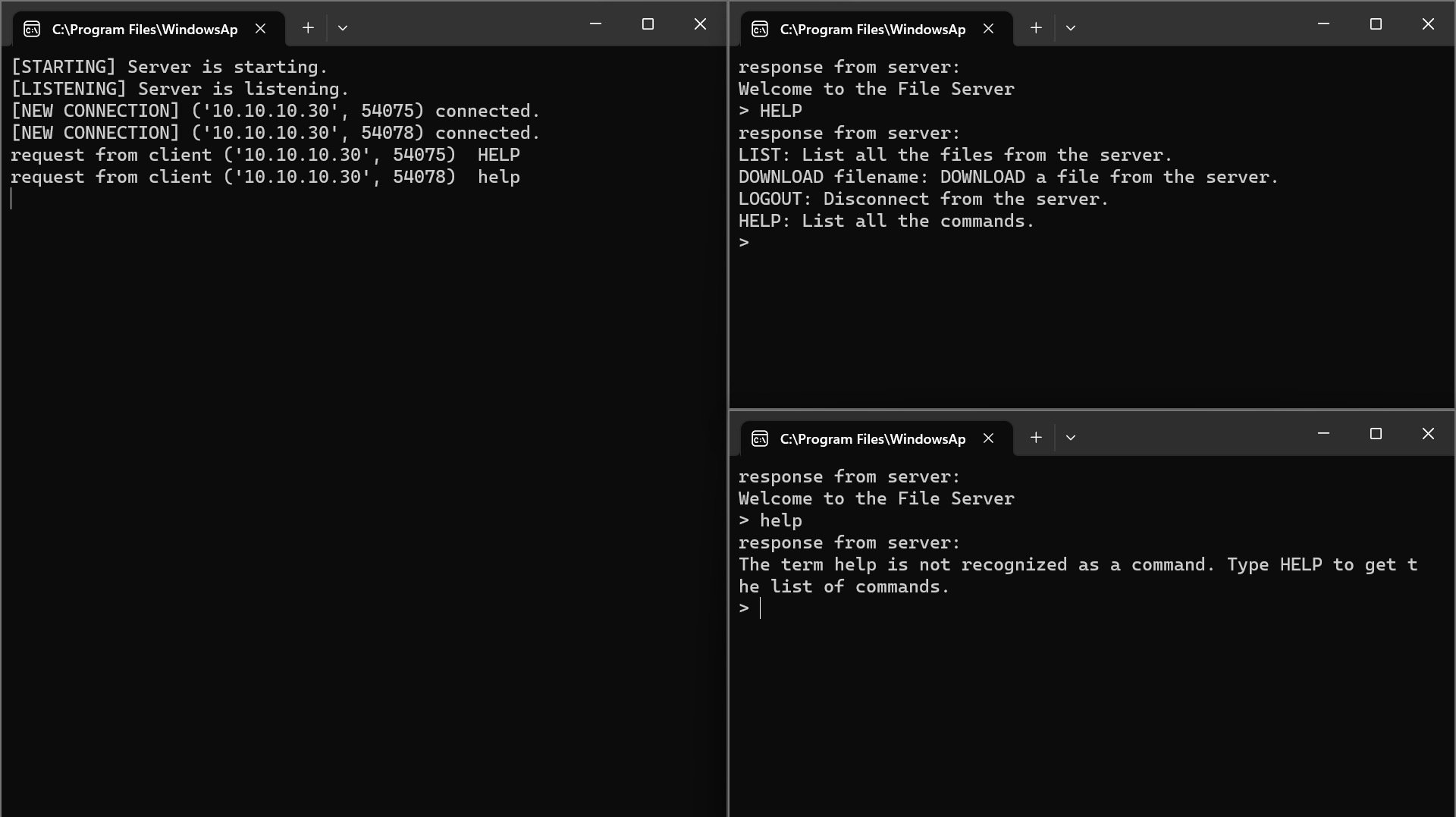
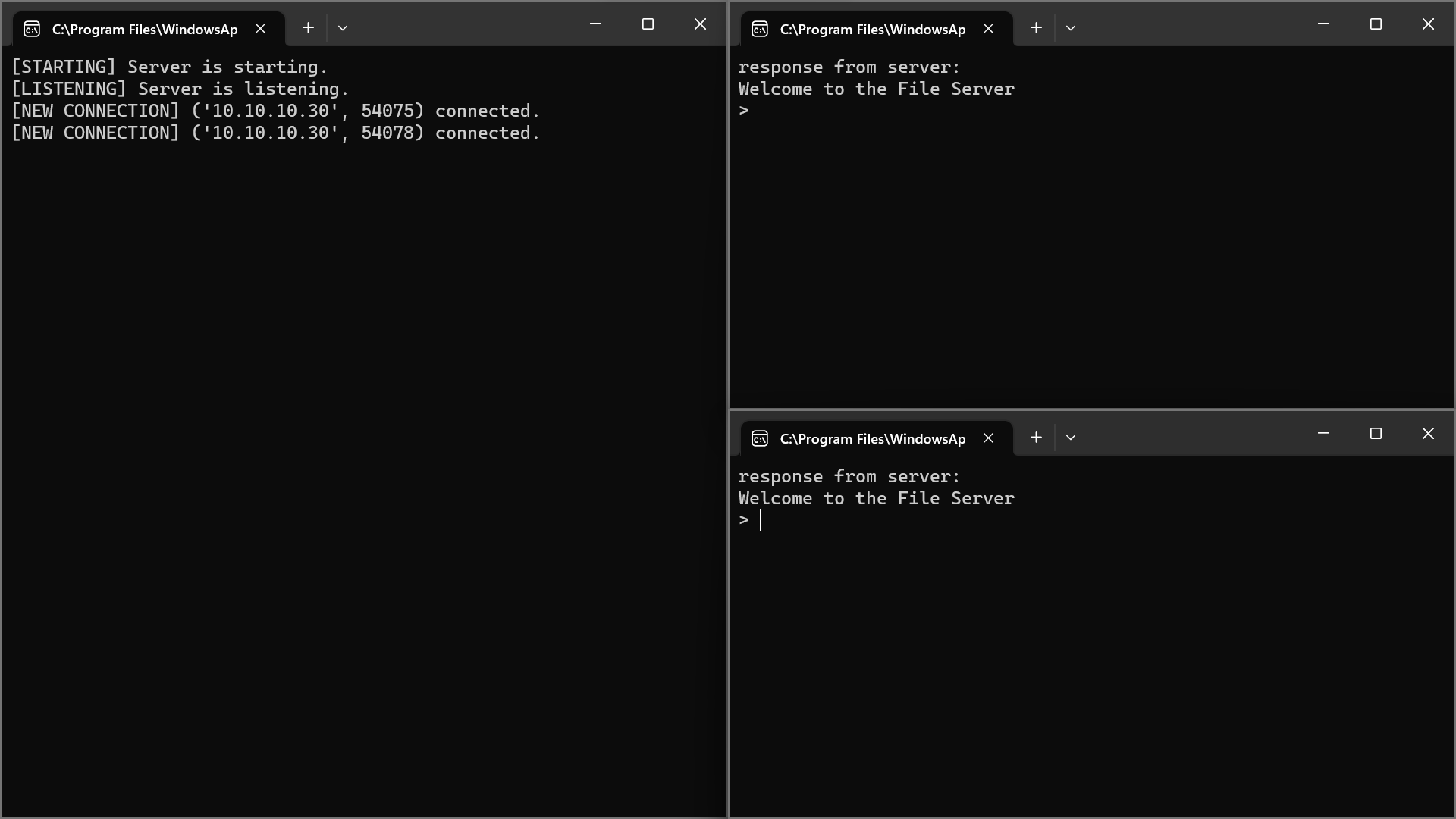
Nous allons montrer ici un cas d’utilisation.

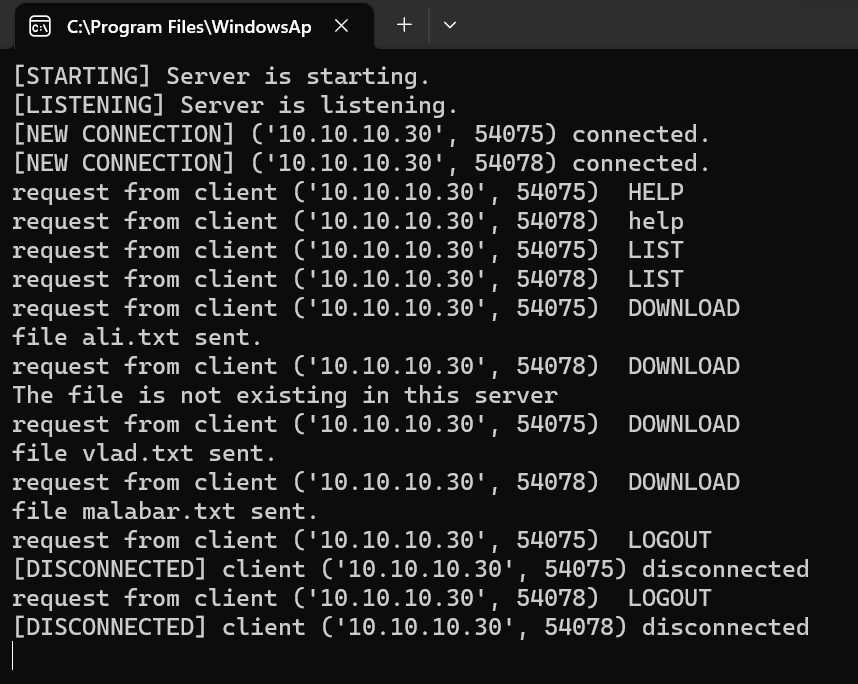
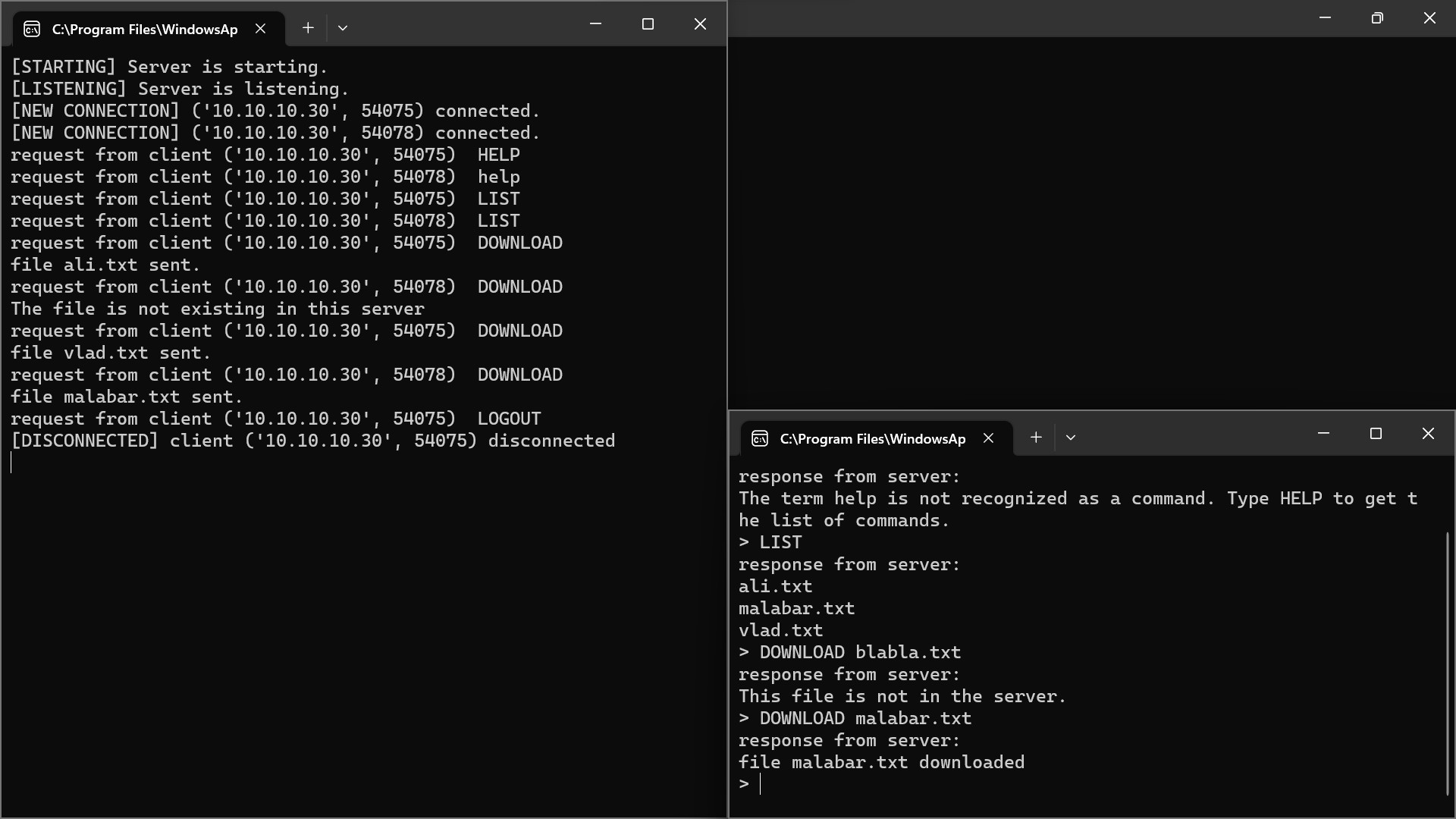
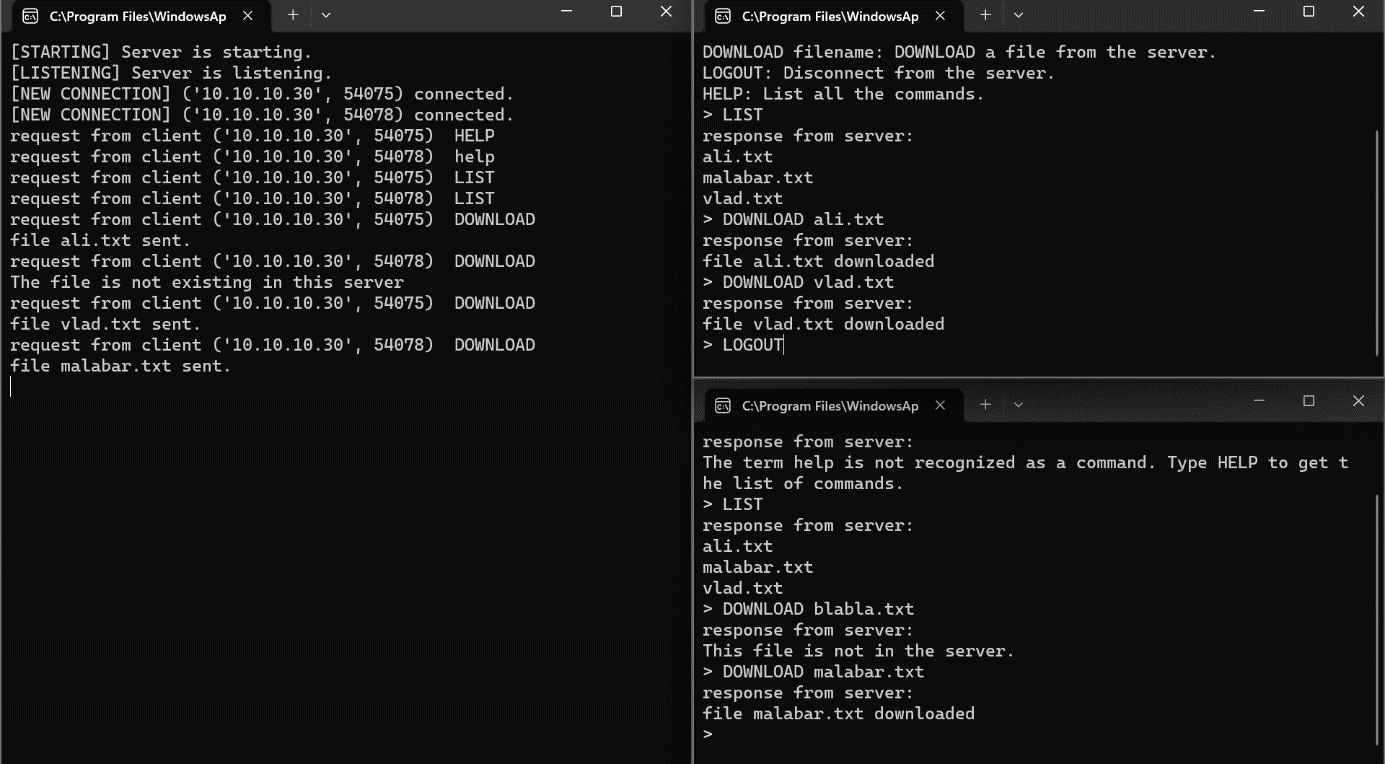
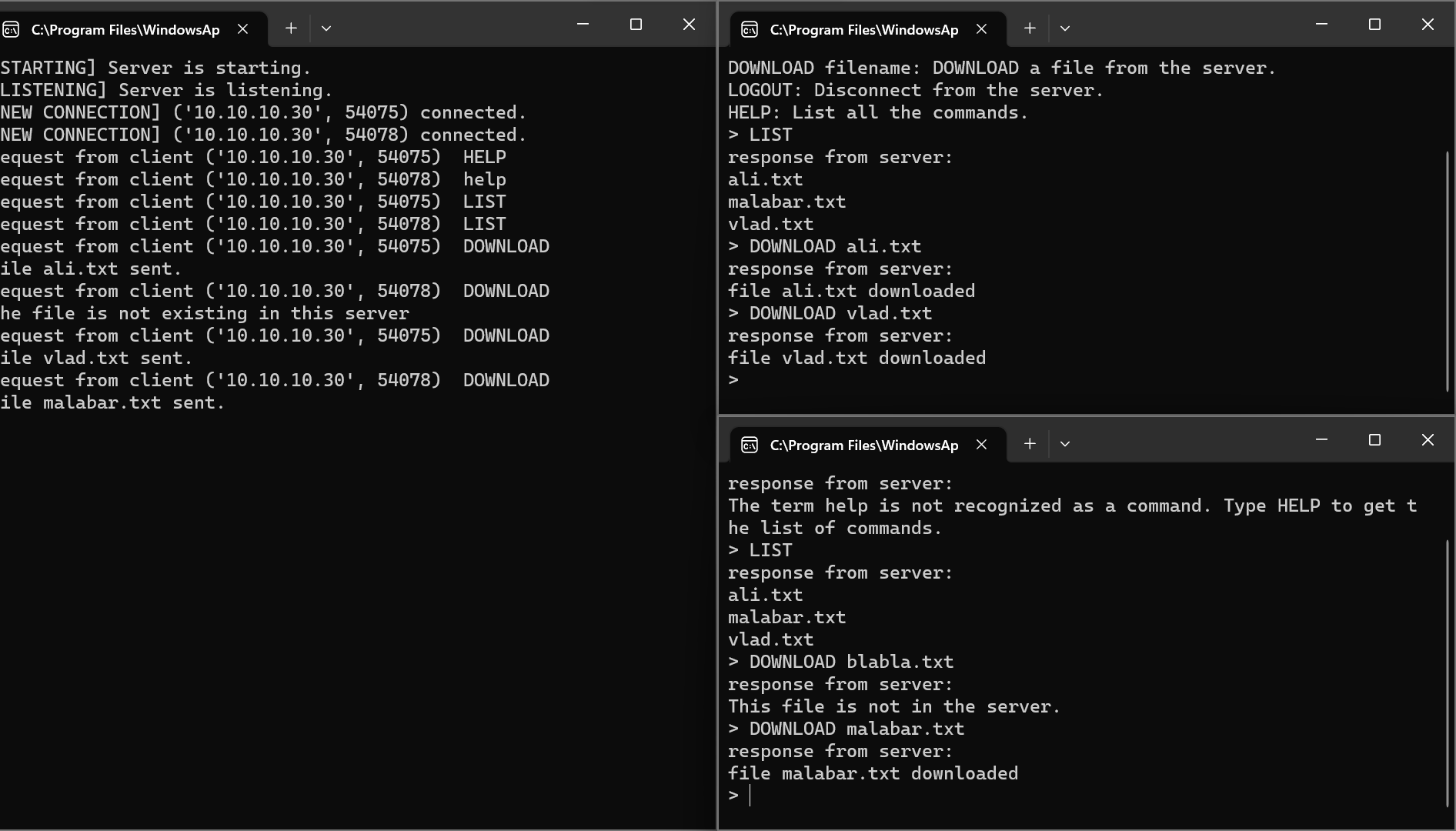
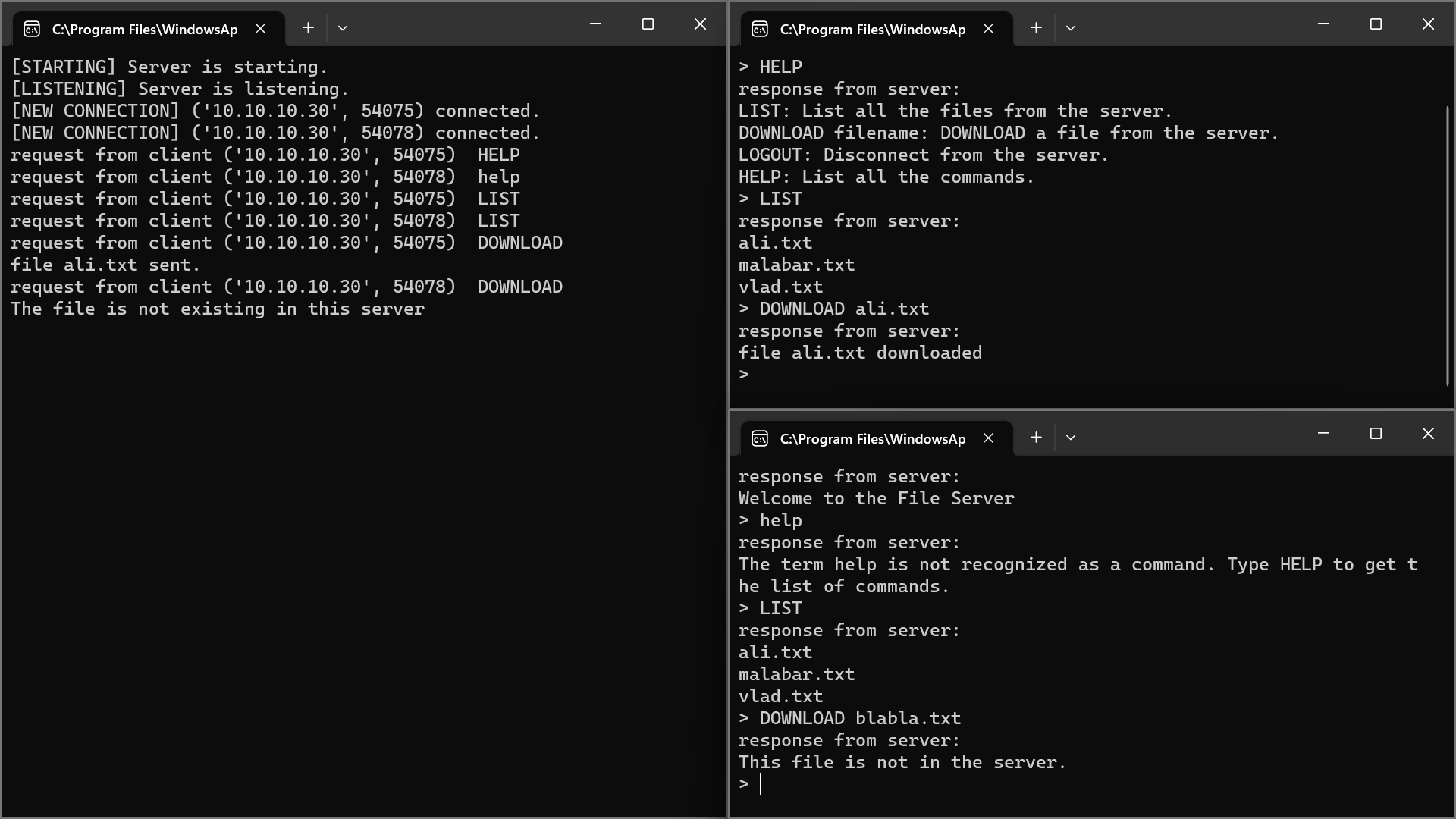
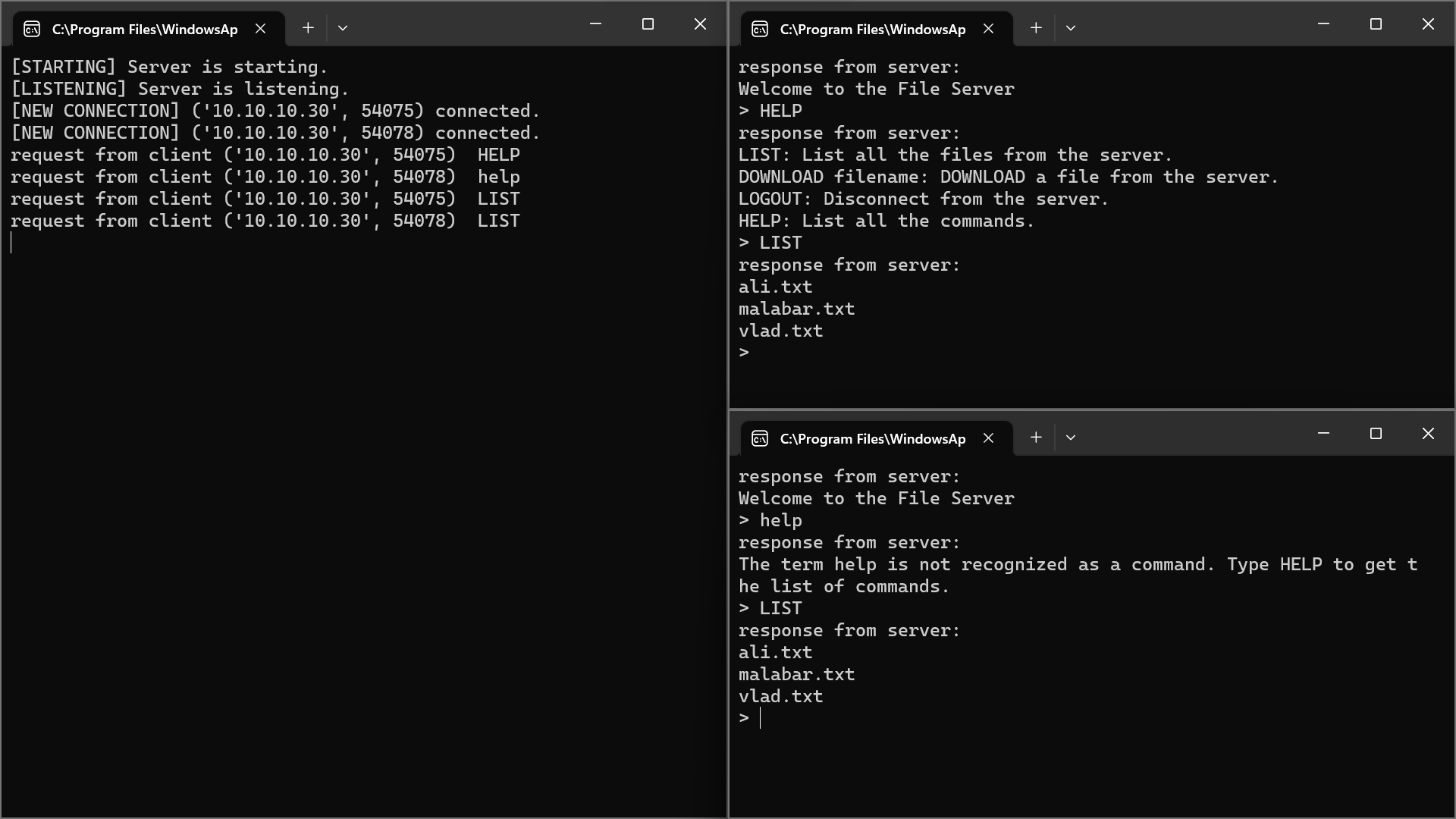
D’abord on lance le serveur qui se met à écouter pour des connexions et ensuite les clients :





Le serveur et les clients lancés on peut démarrer les échanges :





Dans cet exemple les différents cas ont été abordés.

# CONCLUSION

En conclusion le serveur de fichier est bien fonctionnel et on peut modifier différentes options si on désire qu’il fonctionne et réponde de manière spécifique. Le serveur peut traiter plusieurs clients configurés pour communiquer avec lui afin d’avoir accès aux ressources du serveur. Le protocole d’échange entre clients et serveur fonctionne bien et assure une bonne structure entre les messages échangés. On peut implémenter la solution pour que peu importe la position du client il lui suffise de connaitre l’adresse IP et le port d’écoute du serveur pour qu’il puisse accéder au serveur et ses services.