Bienvenue sur la documentation du serveur,

Je vais dans un premier temps, expliquer comment fonctionne le serveur; quelle architecture j'ai utilisé et comment je m'y suis pris.
Ensuite, ce que j'ai fait mais aussi ce que je n'ai pas fais et les raisons pour cela.

Le programme à utiliser sera **S1.py** de mon repository github (ici si jamais vous ne l'avez pas déjà)

- comment est architecturé le serveur :

Le serveur fonctionne en asynchrone avec le client, chacun peut envoyer et recevoir des messages sans attendre une réponse de l'autre, de ce fait, nous pouvons enchaîner les commandes les unes après les autres sans interruption extérieure.

Le fait qu'il soit asynchrone permet de gérer les messages envoyés par le client et de les traiter en conséquence si elles sont reconnues par le programme, si ce n'est pas le cas un message sera envoyé au client comme quoi la commande n'est pas reconnue.

Le serveur crée un socket sur l'adresse de **localhost** sur le **port 5108**, il attend une connexion avant de commencer les threads, il ne peut alors pas envoyer ou recevoir des messages tant qu'un client n'est pas connecté.

Le serveur possède 2 threads, un pour l'envoi de message et un autre pour la réception de message.

Le thread de l'envoi fonctionne de manière à ce que quand le serveur veut envoyer un message du type 'disconnect', le client se déconnecte et le serveur attend une nouvelle connexion, en soit, le serveur n'est pas sensé envoyé de message mais j'ai trouvé plus intéressant de faire de cette manière bien qu'ici cette fonction est inutile car elle ne marche pas de cette façon, en effet, c'est le client qui envoie ces messages et non le serveur.

Le thread de l'envoi, lui, est plus intriguant que celui vu précédemment, il permet de recevoir et décoder le message reçu, tant que le message n'est pas 'kill', 'reset' ou 'disconnect', la communication continue, si c'est le cas alors le serveur ferme la socket et ni le client ni le serveur peut envoyer ou recevoir de message. Si le message correspond à une commande définie dans une fonction, le résultat sera envoyé en réponse de la commande au client, comme précisé dans la documentation du client, les commandes systèmes reconnus sont : ip, ram, os, cpu, hostname. Les messages que le serveur reçoit sont affichés dans le terminal de ce dernier afin de voir ce qui a été réceptionné mais surtout si tout s'est bien passé pendant l'échange, c'est-à-dire si le serveur à bien acquit le message.

- ce qui a été fait et non :

J’ai pu établir une connexion asynchrone, ça n’a pas été facile car j’ai eu quelques difficultés avec la communication avec le serveur et le client, je ne comprenais pas comment je devais faire les threads ou plutôt ce qu’il devait y avoir dans les fonctions à threader, j’ai pu comprendre ce qui n’allait pas en expérimentant et en essayant, malheureusement, j’ai pris peut-être un peu de temps à faire en sorte que ce soit asynchrone ce qui m’a fait perdre du temps sur d’autres tâches que je devais réaliser, mais cela a quand même été utile vu qu’au final ça marche sans soucis.

Ensuite ce sont les commandes que j’ai faites, ça n’a pas été compliqué, les explications de monsieur Drouhin m’ont aidé à comprendre comment les messages devaient agir sur la socket, de ce fait, j’ai interprété les commandes dans les threads du serveur afin que quand le serveur les reçoit, il agit en conséquence.

J’ai aussi fait les interprétations de quelques commandes systèmes, ce n’était pas compliqué en soit mais certaines ont été plus compliqué à élaborer que d’autres, ce n’était pas facile de trouvé les bons modules capables de montrer ce qui devait l’être, seulement, ils sont uniquement effectifs sous windows et non sous linux ou Mac (bien que c’est possible qu’elles marchent quand même) et par conséquent, je n’ai pas pu remplir correctement le cahier des charges, les actions comme ‘ping’ ou ‘dos: …’ sont inopérationnelles à cause notamment d’un manque de temps mentionné plus haut, c’est donc cette partie que je n’ai pas réussi.

Après les commandes que je n’ai pas pu faire, il y la gestion des nouvelles machines dans un fichier csv que je n’ai pas fait, si cette aspect là n’est pas présent dans mon projet, c’est parce que je voulais me concentrer sur les choses plus importantes comme le fonctionnement du serveur et du client afin qu’il puissent communiquer sans problème d’une manière asynchrone mais aussi sur l’interface graphique du client, que j’ai mis un peu de temps à faire car je ne saisissais pas comment lier la socket avec cette dernière, puis avec un peu d’aide, j’ai essayé à ce que ce soit pertinent d’un point de vue pratique car si il y a une interface graphique, il faut qu’il ait le résultat de la commande qui s’affiche sur le terminal mais aussi la gestion des erreurs.