

Document de réponse

Introduction :

Ce projet, réalisé dans le cadre de la SAE 3.02, avait pour objectif de concevoir une architecture Clients/Serveurs, permettant l'exécution de programmes soumis par des clients. Le système repose sur une architecture multi-serveurs, avec un serveur maître chargé de gérer les clients et de répartir les tâches entre plusieurs serveurs secondaires pour assurer une meilleure répartition de la charge.

L'accent a été mis sur plusieurs aspects essentiels, notamment :

- La communication client-serveur via sockets.
- Le développement d'interfaces utilisateur avec **PyQt**.
- La gestion de la charge des serveurs secondaires grâce à un mécanisme de **load balancing**.
- La prise en charge de différents langages de programmation pour l'exécution des programmes soumis (Java, Python, C, C++).

Fonctionnalités Implémenté :

- Client :

Script de Base **client.py** :

Ce script, grâce à la programmation orientée objet, permet d'initialiser un ou plusieurs clients en utilisant des arguments tels que le port ou l'IP du serveur maître.

Il permet d'utiliser toutes les fonctions de base et essentielles pour que le client fonctionne, telles que :

- **connect** : pour se connecter au serveur,
- **envoi** : pour envoyer un message,
- **reçois** : pour recevoir des messages et les afficher, etc.

Interface PyQt **interface_client.py** :

Cette interface, créée avec le module PyQt de Python, permet d'utiliser les fonctions du script de base via des boutons cliquables.

Elle offre également la possibilité de choisir un fichier, de l'afficher, de le modifier si nécessaire, puis de l'envoyer.

Elle permet également d'afficher les logs en cas d'informations importantes ou d'erreurs, et surtout d'afficher les messages de résultats du serveur.

- Serveur :

Script de Base `server.py` :

Le script de base du serveur a pour rôle d'exécuter les tâches essentielles du serveur, à l'exception de la connexion. Au lieu de cela, il accepte les connexions des clients et des serveurs secondaires.

Il sert de pont entre les clients et les serveurs secondaires.

En outre, il gère le **load balancing** des messages, afin d'envoyer les tâches à exécuter sur un serveur secondaire disponible.

Gestion des serveurs secondaires :

La façon dont mon programme et mon infrastructure sont conçus me permet d'utiliser les serveurs de deux manières différentes :

1. **Multi-processing sur la même machine** : Cela permet de lancer automatiquement les serveurs secondaires à l'aide de `subprocess`.
2. **Lancement manuel sur des machines distantes** : Cette option peut être désactivée, permettant de lancer les serveurs secondaires manuellement sur d'autres machines.

Interface PyQT `interface_server.py` :

Cette interface graphique permet de configurer le serveur maître en lui attribuant les ports d'écoute client et serveur.

Elle permet également d'activer ou de désactiver le démarrage des serveurs secondaires localement et de les configurer si nécessaire.

Elle offre aussi la possibilité d'allumer ou d'éteindre le serveur maître (notez qu'en éteignant le serveur maître, cela éteint également les serveurs secondaires).

Enfin, elle permet de visualiser les serveurs secondaires connectés et de vérifier leur disponibilité.

- Serveur Secondaire :

Script `serversecondaire.py` :

Le script de base du serveur secondaire a pour rôle de recevoir les tâches envoyées par le serveur maître, puis de les traiter et les exécuter en fonction du langage de programmation.

Le serveur secondaire possède une limite de tâches et de CPU, définie par l'utilisateur lors du lancement.

Si le serveur reçoit une tâche et que sa capacité en CPU ou le nombre de tâches en cours ont atteint leur limite, il n'accepte pas la tâche et renvoie cette dernière au serveur maître en spécifiant qu'il n'est pour l'instant pas disponible.

Lorsque qu'une tâche dans la file d'attente est terminée, le serveur secondaire renvoie le résultat de l'exécution au serveur maître, qui se chargera de le renvoyer au client.

Une limite de temps de 3min pour l'exécution d'un script est mise en place pour éviter de bloquer le serveur dans une boucle infini.

Fonctionnalités Non Implémenté :

- **Sauvegarde des Tâches :**

- Les tâches en cours ne sont pas sauvegardées en cas de panne ou d'interruption du serveur. Une solution de persistance des tâches était prévue mais n'a pas été implémentée en raison de contraintes de temps.

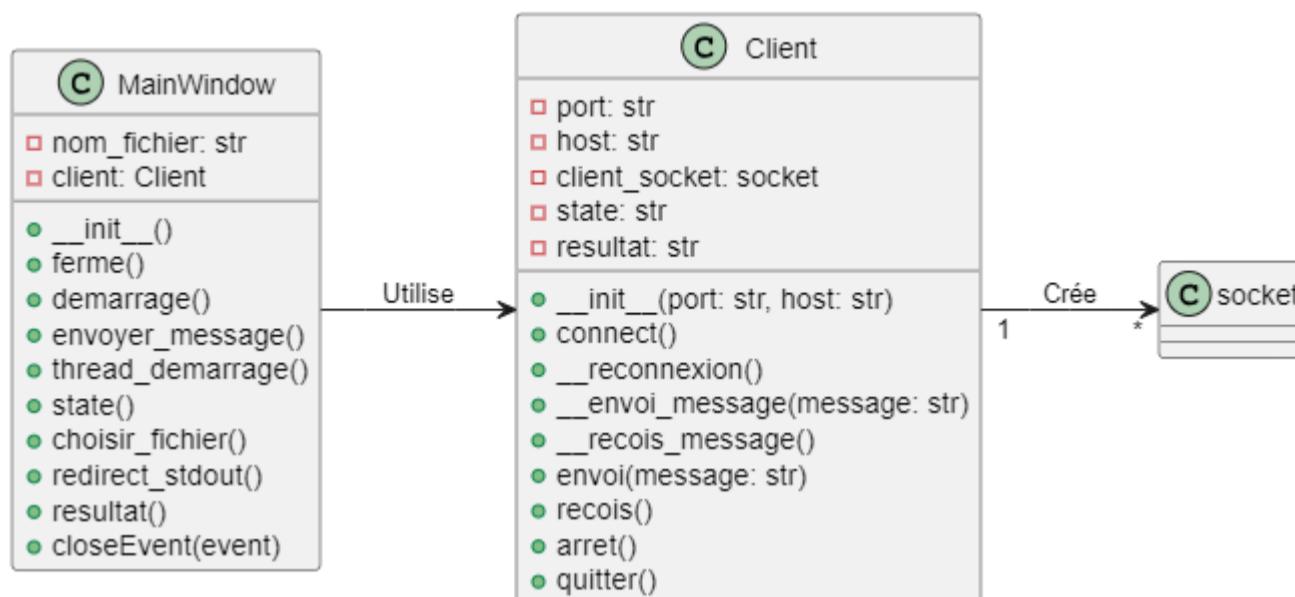
- **Sécurité (Cryptage des Communications) :**

- Le cryptage des données échangées entre les différents composants du système n'a pas été mis en place, bien qu'il ait été envisagé.

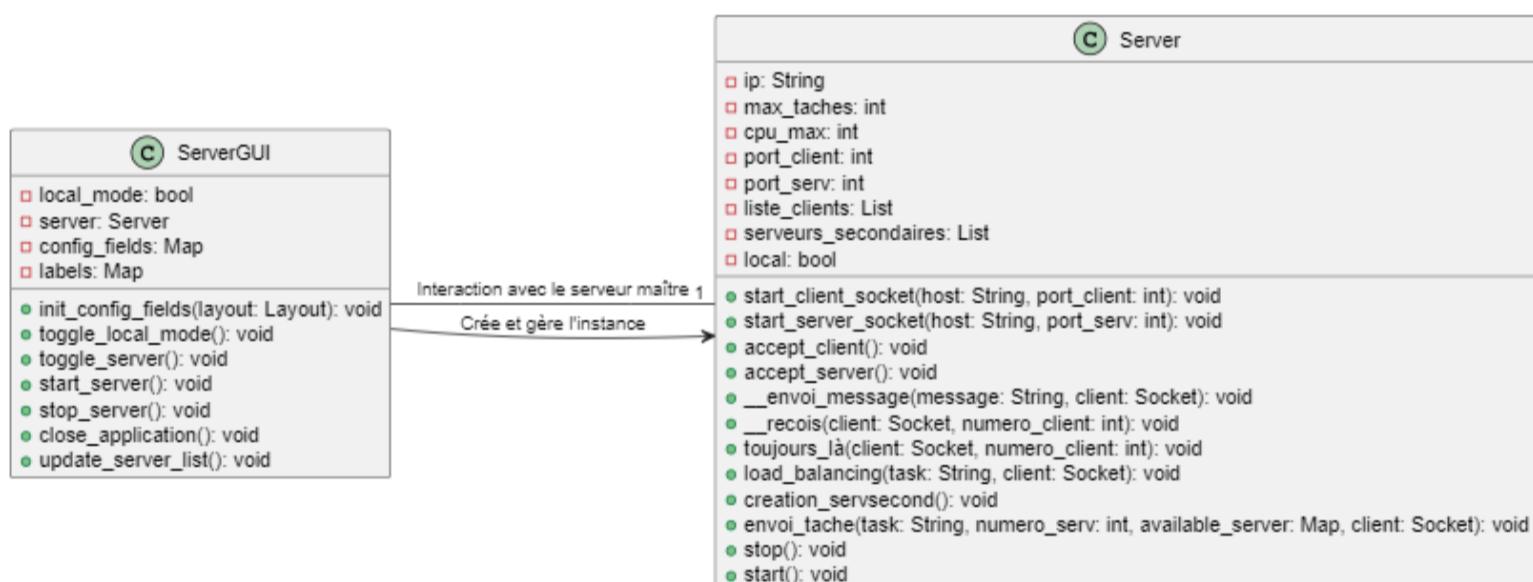
Architecture :

Du code

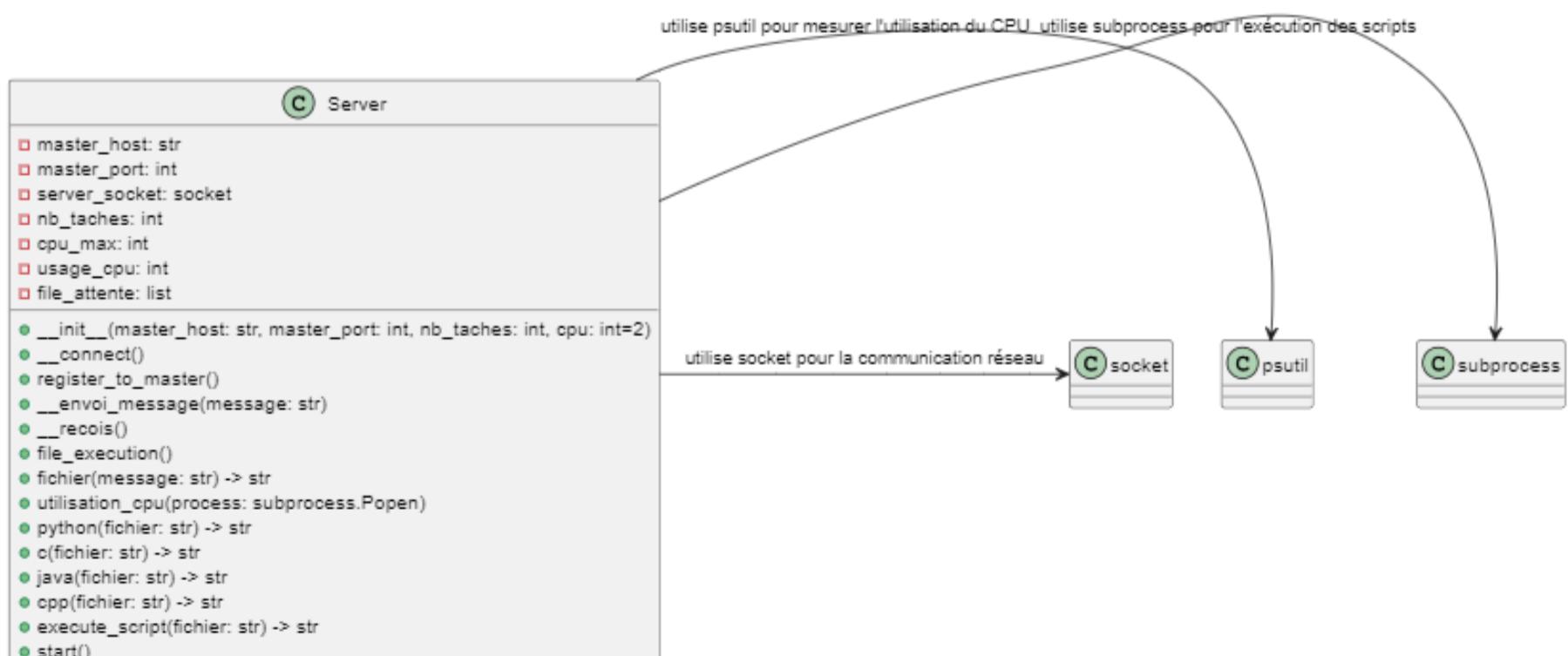
- Diagramme UML du client et son interface :



- Diagramme UML du serveur et son interface :



- Diagramme UML du client serveur secondaire :



- **Structure du projet :**

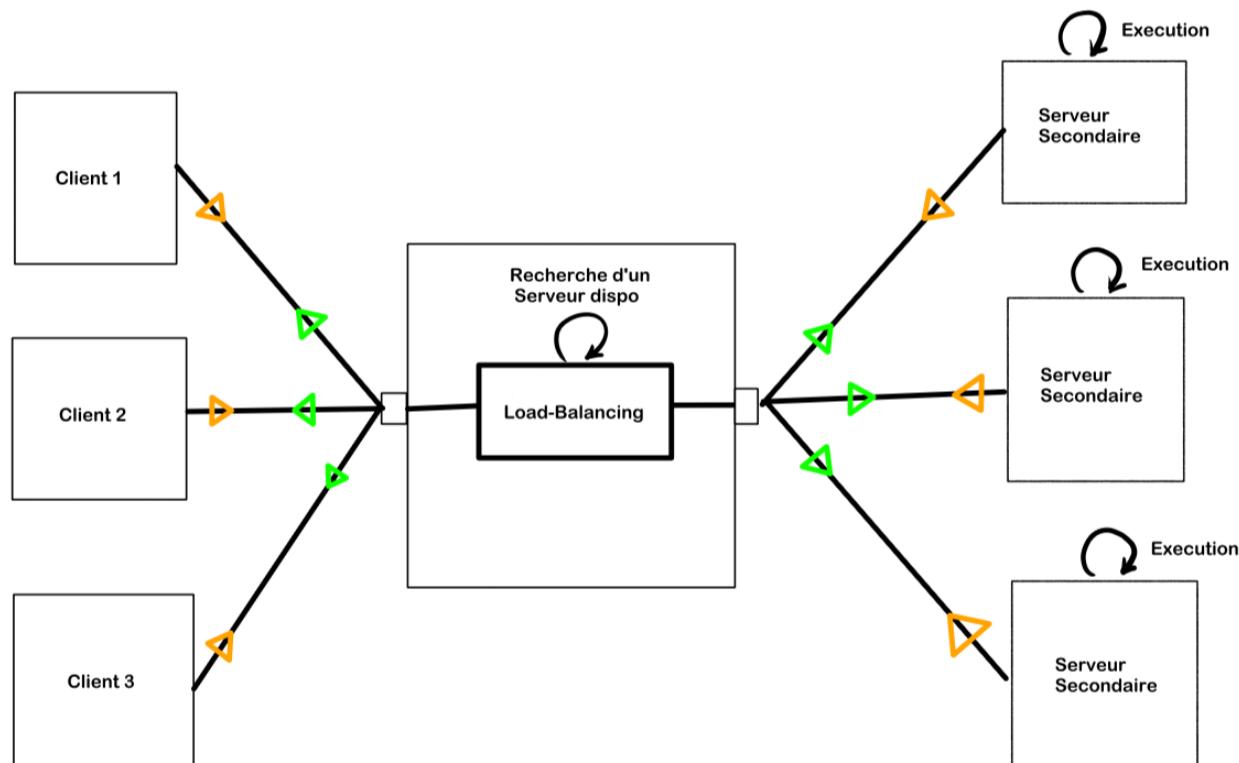
```

SAE_3.02/
|
├── client.py
├── Interface_client.py
├── interface_server.py
├── server_second.py
├── server.py
└── style.qss
|
├── execution/
│   └── sert lors de l'execution des taches
|
└── test/
    └── contient des fichiers à envoyer pour effectuer des tests

```

De l'infrastructure

- schéma



Planning :

Étape	Durée estimée	Période	Statut
Conception d'un client et d'un serveur non graphique	1 semaine	11 novembre - 17 novembre	<input checked="" type="checkbox"/>
Conception de l'interface graphique du client	1 semaine	18 novembre - 24 novembre	<input checked="" type="checkbox"/>
Conception des serveurs secondaires et transfert des messages	1 semaine	25 novembre - 1er décembre	<input checked="" type="checkbox"/>
Exécution de code (Java, Python, C, C++)	3 jours	2 décembre - 4 décembre	<input checked="" type="checkbox"/>
Mise en place du load balancing	1 semaine	5 décembre - 11 décembre	<input checked="" type="checkbox"/>
Refonte de la communication serveur maître-secondaire	1 semaine	12 décembre - 18 décembre	<input checked="" type="checkbox"/>

Lancement automatique des serveurs secondaires via subprocess	3 jours	19 décembre - 21 décembre	<input checked="" type="checkbox"/>
Tests, gestion des erreurs et correction de bugs	4 jours	22 décembre - 26 décembre	<input checked="" type="checkbox"/>
Rédaction des rapports et réalisation de la vidéo	jour	26 décembre - 30 décembre	<input checked="" type="checkbox"/>

Sécurité :

Failles Identifiées

- **Manque de cryptage des communications** : Les échanges entre le client, le serveur principal et les serveurs secondaires ne sont pas sécurisés, ce qui pourrait exposer les données à des attaques de type "man-in-the-middle".
- **Injection de programme malvaillant** : Les programmes envoyés par le client ne sont pas vérifiés avant leur exécution, ce qui peut constituer une porte d'entrée pour des attaques par injection.
- **Possibilité de falsification du serveur secondaire** : Le serveur secondaire n'est pas authentifié, ce qui laisse la porte ouverte à une usurpation d'identité. Un attaquant pourrait se faire passer pour un serveur secondaire légitime et compromettre le système.

Solutions Proposées

- **Cryptage SSL** : Mettre en place un cryptage des communications via SSL pour sécuriser les échanges entre les différents composants.
- **Validation des programmes** : Ajouter un système de validation pour vérifier les programmes envoyés par les clients avant leur exécution, afin d'éviter l'exécution de codes malveillants.
- **Authentification** : Mettre en place une authentification pour les serveurs et les clients.

Conclusion :

Temps Passé

Le projet a nécessité environ 60 heures de travail, ce qui correspond aux prévisions initiales. Cependant, certaines fonctionnalités optionnelles n'ont pas été réalisées en raison du manque de temps.

Enseignements

Ce projet a permis de renforcer mes compétences en programmation réseau et de mieux comprendre la communication client-serveur ainsi que le fonctionnement d'un cluster de serveurs. J'ai également découvert et appliqué la notion de **load balancing**.

Cette SAE m'a également permis de découvrir **PyQt** pour le développement d'interfaces graphiques, un module que je réutiliserai sans aucun doute à l'avenir..

Enfin, j'ai pris conscience de l'importance de la gestion des erreurs et de la sécurité dans un environnement client-serveur, des éléments essentiels pour garantir la stabilité et la fiabilité d'un système.

Annexes :

- Lien vers le GitHub : <https://github.com/Sylverthorn/GIT-R309-SAE302>