Rapport final SAE 3.02 :

Table des matières :

Rapport final SAE 3.02 : 1

Introduction : 2

Description de la SAE : 2

Schéma de mon Projet : 2

Clients : 3

Interface graphique : 3

Fonction de récupération et modification de fichier : 3

Mise en place de sécurité : 4

Terminale de suivi : 4

Master : 5

Interface graphique : 5

Transfert des données : 5

LoadBalancing : 5

Slaves : 6

Récupération des fichiers : 6

Compilation et exécution : 6

Interfaces graphiques : 6

Problèmes rencontrés : 7

Améliorations possibles : 7

Conclusion : 7

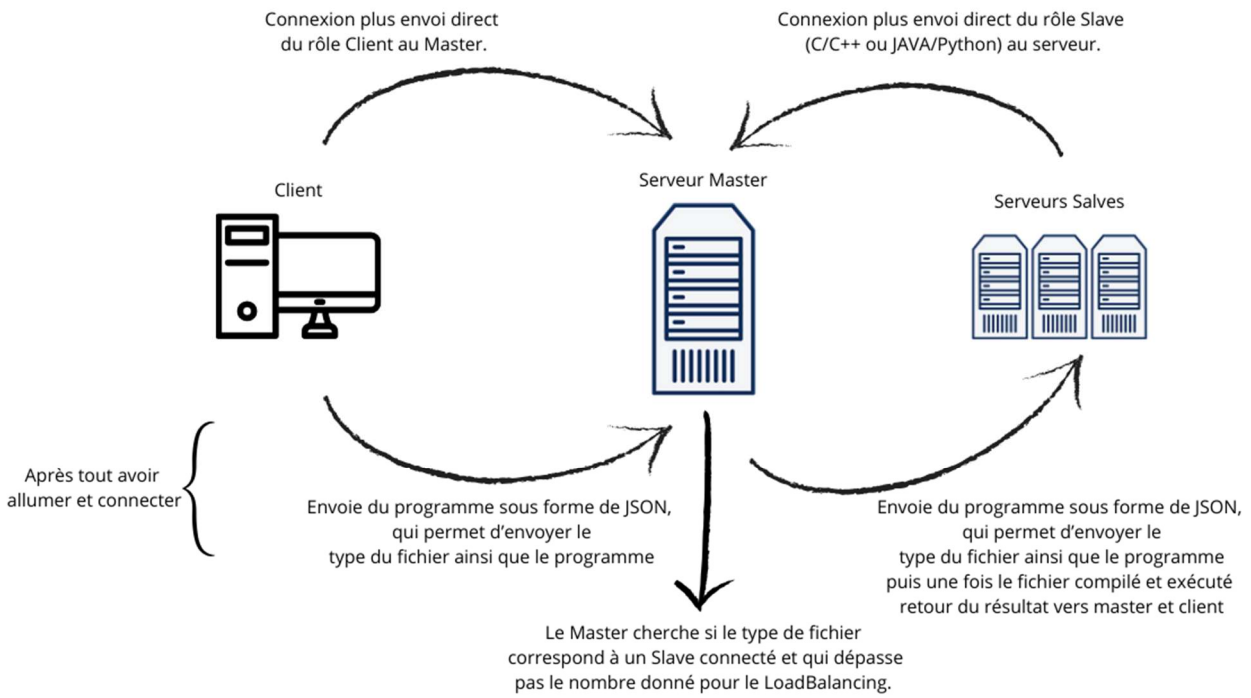
Introduction :

Description de la SAE :

Le but de ce projet est de créer et mettre en place un système multiserveurs capable de recevoir des demandes de clients, de compiler et d'exécuter des programmes dans un langage spécifique, tout en répartissant automatiquement le travail entre plusieurs serveurs. Ce système doit également pouvoir gérer plusieurs clients en même temps.

Schéma de mon Projet :

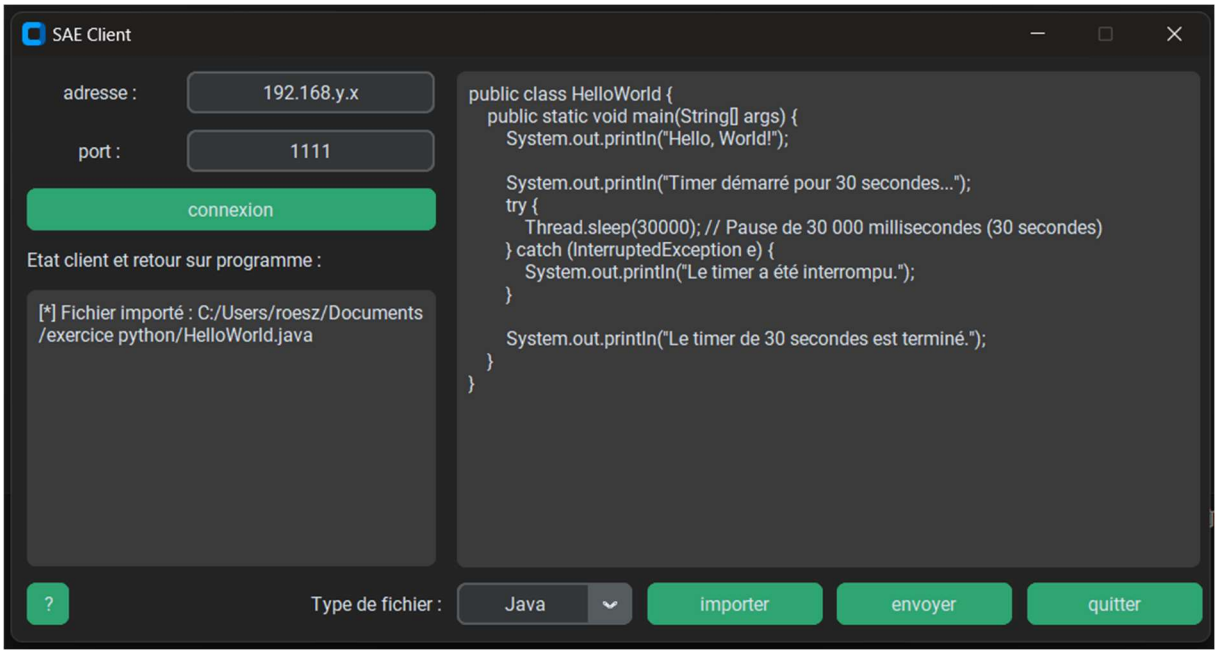
Comme vous pouvez le voir, j'ai essayé de mettre mon projet sous forme de schéma, on retrouve donc la définition des rôles dès le lancement des clients et slaves. Ces rôles sont récupérés par le Master (ce qui permet de différencier les sockets client car oui les slaves sont techniquement des clients du master). Ensuite on peut directement envoyer depuis le client, le socket va passer par le master qui va l'envoyer au bon slave selon sont rôle et le type de fichier à compiler et exécuter.



Clients :

Interface graphique :

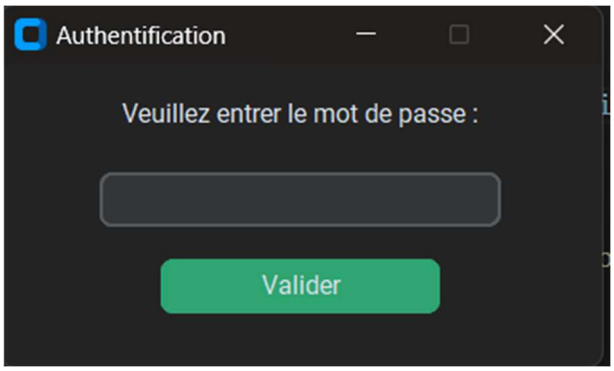
L'interface graphique a été faite à l'aide de la librairie CustomTkinter qui a été autorisé suite à une demande de ma part en début de SAE. Elle permet de gérer la connexion vers le serveur Master, de directement sélectionner un fichier depuis l'explorateur de fichier et de le modifier a vôtre convenance avant d'envoyer le fichier au Master. On peut aussi retrouver en bas a gauche de l'interface un suivi des logs (messages de connexion, de retour du programme ou d'erreurs). On retrouve aussi un bouton d'aide qui vous redirige directement sur ma page GitHub de la SAE en cas de besoin.



Fonction de récupération et modification de fichier :

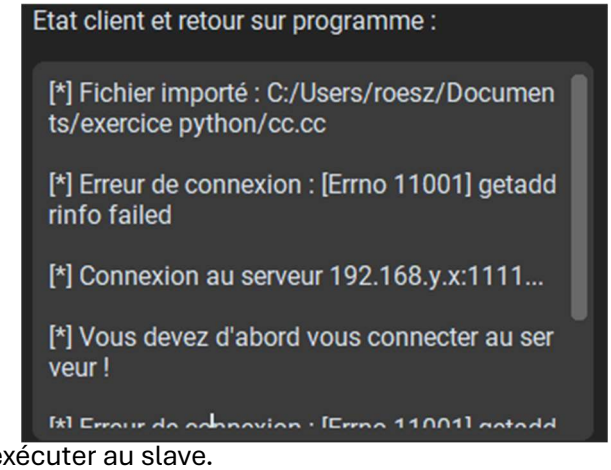
Pour la récupération de fichier, j'ai mis en place un bloc ComboBox qui permet de choisir le type de fichier qu'on veut rechercher dans son explorateur de fichier et envoyer. On a donc le choix dans ce bloc de Sélectionner des fichiers en Python (.py), en Java (.java), en C (.c) ou bien encore en C++ (.cpp, .cc). Pour récupérer un fichier souhaité, on sélectionne donc le type de fichier dans le ComboBox et on fait importer, une fois le fichier sélectionné il apparait dans le bloc en haut a droite qui permet de faire quelques dernières modifications en cas de besoin.

Mise en place de sécurité :



Pour la Sécurité, j'ai mis en place une petite page d'interface graphique avant d'accéder à la page principale client où on doit saisir un mot de passe (Admin). Cela permet de mettre une petite couche supplémentaire pour sécuriser le programme. Pour le code, j'ai fait en sorte qu'on récupère le mot de passe saisie dans l'interface graphique, qu'on l'encode en MD5 et qu'on vérifie si l'encodage est similaire à celui déjà inscrit dans mon code. Dès que le mot de passe correct est saisi, on ouvre la page principale du client. En cas de mauvais mot de passe saisie, la page reste ouverte en attente d'un nouvel essai.

Terminale de suivi :

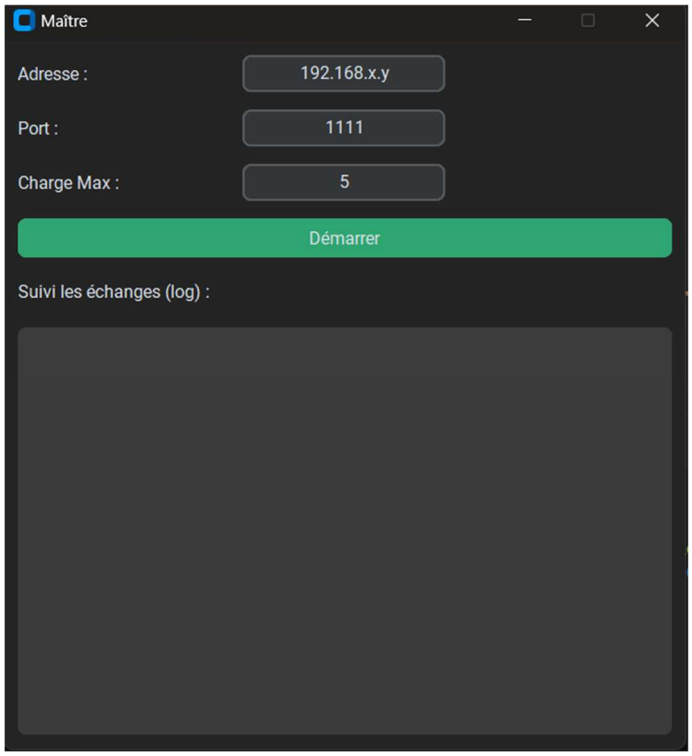


La terminale de Suivi sert simplement d'affichage rapide pour tout ce qui est log j'ai donc ajouter une fonction log qui retourne toute les erreurs et Print('') qui aurait dû être normalement dans le terminale directement dans l'interface graphique pour avoir un meilleur aperçu et une meilleure visibilité des problèmes, on retrouve donc des messages d'erreurs, les tentatives / problèmes liées a la connexion au serveurs, un message pour dire qu'on a importé un fichier et bien sûr le message de retour du programme qu'on a fait exécuter au slave.

Master :

Interface graphique :

Pour une meilleure gestion et une mise en place plus simple j'ai ajouté une interface graphique simple au master. Celle-ci permet de setup l'adresse du serveur ainsi que son port et de mettre en place un nombre de programme max à tourner pour les slaves. Il y a aussi un système de log pour voir les échanges qui se font entre les clients et slaves, les erreurs et connexions ou déconnexion.



Transfert des données :

Pour le transfert et la récupération des données entre les différents rôles, j'utilise des fichiers JSON qui sont écrit sous forme (Type_de_fichier : ... Code : ...) Ce qui me permet de récupérer le type du programme et de l'associer au Rôle du slave correspondant. Les deux types de slaves sont C/C++ et JAVA/Python, ces rôles sont envoyés par les slaves dès lors de la connexion, Le master enregistre cette connexion de rôle dans un tableau. C'est avec cette manière que je peux définir si un slave est connecté ou non et sur lequel envoyer le Programme C par exemple.

LoadBalancing :

Pour le LoadBalancing, je mets en place une récupération de la variable « tache_max » lors du démarrage du serveur. Mon LoadBalancing se fait par nombre de programme traité par type de fichier sur un slave. C'est-à-dire que même un slave qui gère C et C++ avec un nombre de tâche max de 2 pourra exécuter deux fichier C et deux fichier C++. En réalité le Master compte le nombre de type de fichier entrante sur un slave et change de slave si celui-ci dépasse le nombre souhaité.

Slaves :

Récupération des fichiers :

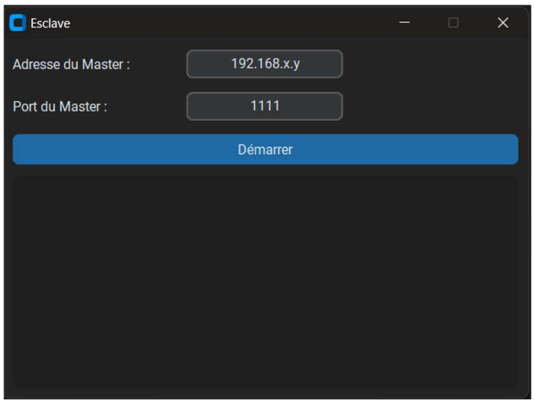
Comme Pour le master, c'est le JSON qui me sert pour récupérer le programme. Sachant que j'ai deux types de slaves (C/C++ et JAVA/Python) j'ai dû récupérer encore une fois le type de fichier dans le fichier JSON, par exemple pour mon slave C/C++ je récupère « C » et je mets donc le code (deuxième argument du JSON) dans la variable qui va passer dans la compilation et exécution du C. l'erreur ou le retour est directement renvoyer au serveur Master.

Compilation et exécution :

Le Problème principale pour la compilation et l'exécution était de Retrouver la class publique en Java pour le nom du fichier. J'ai donc importé le Regex qui m'as permis de récupérer dans Mon fichier JSON le mot après la suite de Caractère « Public Class » qui m'a servi pour le nom de l'exécutable. Pour compiler j'ai dû installer des paquets qui seront écris dans ma Doc d'installation et de mise en place sur mon repository GitHub.

Interfaces graphiques :

Pour simplifier la logique de mon projet j'ai décidé de mettre mes Slaves sous forme de clients socket, au lieu de faire en sorte que le master se connecte au slave. Dans mon programme les slaves sont donc des clients du Master ce qui fait qu'ils doivent s'y connecter. J'ai donc décider d'ajouter une interface graphique à mes slaves pour pouvoir saisir plus facilement l'adresse et le port et avoir un meilleur suivi de ce qu'il se passe sur mes slaves. Cette partie n'était pas spécifié dans le Cahier des charges mais le semblait plus pratique pour la mise en place et pour facilement suivre le LoadBalancing



Problèmes rencontrés :

Pour cette SAE, j'ai eu tout d'abord des problèmes pour la logique derrière la connexion entre un Master et des clients et slaves (socket client) car je pensais d'abord les différencier avec une adresse IP. Jusqu'à avoir l'idée des rôles qui m'as permis de beaucoup simplifier la connexion et l'identification de chacun.

Le second problème était quant à lui lié à l'échange des fichiers et la reconnaissance du type de fichier. J'essayais d'abord d'envoyer directement un fichier intitulé « Fichier.py » par exemple pour du python et de reconnaitre directement chez qui il devait aller, mais voulant ajouter le LoadBalancing à envoyer au slave je me suis dit que c'était trop de message à envoyer donc je suis passer sur des fichiers JSON qui permettent directement de mettre ce qu'on veut en arguments j'ai vite réussi a le mettre en place et j'ai garder ça comme tel même si finalement j'ai mis la gestion du LoadBalancing sur mon Master.

Le dernier problème rencontré était la récupération du nom de fichier pour l'exécutable JAVA mais c'est finalement passer avec quelques essaie avec du Regex

Améliorations possibles :

Pour Améliorer mon programme j'aurais pu mettre un système plus complexe pour le LoadBalancing avec une charge de CPU à respecter. Actuellement mon LoadBalancing réside juste dans un nombre de programme à respecter par type de fichier.

Pour une meilleure sécurité j'aurais pu encoder chaque échange client-master et master-slave pour éviter que les messages circulent en clair sur le réseau.

On aurait pu aussi mettre en place un système où au moins un slave pour chaque type de programme se lance directement dès la connexion au serveur mais j'aurais donc du mettre des slaves directement sur la VM Master qui tourne en parallèle ou bien trouver une alternative pour une VM distante.

Conclusion :

Pour Conclure, ce projet m'as permis d'améliorer mes compétences en mise en place de client – serveur sur python, d'apprendre à compiler et exécuter des programmes directement en python et de m'améliorer globalement sur plein d'aspect (thread, interface graphique, socket) ça m'a aussi permis de mieux visualiser le nombre de failles possibles pour des infrastructures comme celle-ci. La sécurité mise en place est bien trop légère et peux être compromise très rapidement avec un simple brut-force, même si bien sûr je n'aurais pas mis Admin comme mot de passe pour une vrai application.