Rapport Final SAE 3.02

Table des matières

Introduction :

Pour ce projet, nous avions deux choix :

- Le sujet initial, qui consiste à créer un serveur maître redirigeant les programmes des utilisateurs vers un serveur esclave. Ce dernier exécute le programme, renvoie le résultat au serveur maître, qui à son tour le transmet à l'utilisateur. Ce sujet est noté sur 40, avec une note ramenée sur 20.
- Le sujet simplifié, qui consiste en un serveur unique interagissant directement avec un utilisateur. L'utilisateur envoie un programme, le serveur l'exécute, puis renvoie le résultat. Ce sujet est noté sur 20.

Dans un souci de gestion de temps j'ai opté pour le sujet simplifier

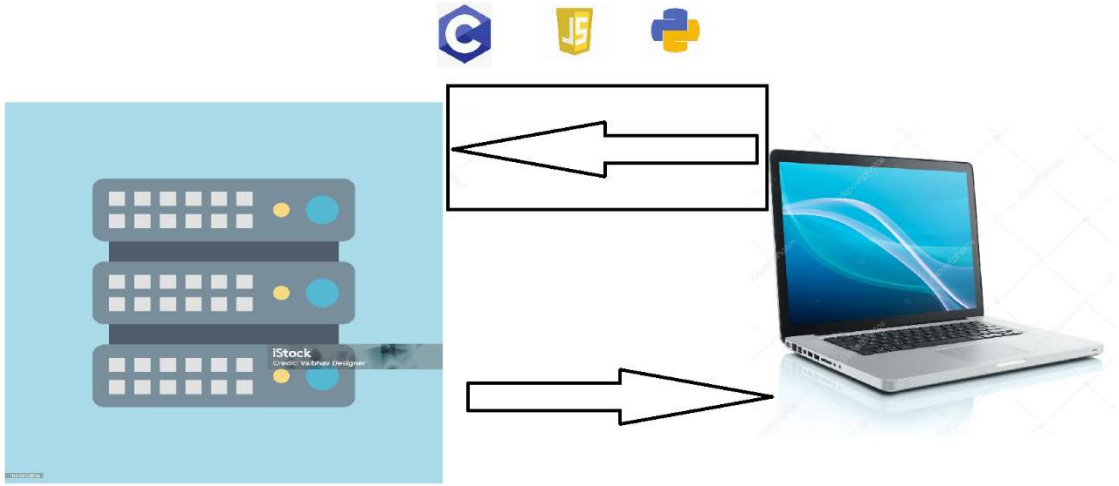
Mise en place :

Architecture globale :

Cette architecture est composée de deux machines (minimum) :

- Client : Cette machine permet de se connecter au serveur d'exécution via une interface graphique. L'utilisateur saisie l'IP du serveur, sélectionne le programme à exécuter et l'extension du programme. La machine client réceptionne le résultat du programme envoyer par le serveur (sortie standard ou sortie d'erreur) et l'affichera dans l'interface graphique.
- Serveur : Ce dernier reçoit les connexions des clients, exécute le programme envoyé, et retourne le résultat. Il ne permet qu'une exécution de programme à la fois. Les clients qui envoyant un programme lorsqu'un programme est en cours d'exécution sont rejetés et les clients son inviter à se connecter à un autre serveur.

Transfert du programme a exécuter par le serveur (Pthon, Javscript ,C)



Envoie de la réponse ou de l'erreur générer par le programme

Choix des technologies :

Langages de programmation

Le projet a été entièrement programmé en Python. L'exécution des différents programmes sur le serveur nécessite également l'utilisation de commandes Bash.

Bibliothèques

Voici les bibliothèques utilisées et leur rôle :

- **Socket** : Assure la transmission des données entre le serveur et le client.
- **Datetime** : Fournit la date et l'heure actuelles, permettant de mesurer la durée d'exécution sans utiliser la bibliothèque *time*.
- **Threading** : Permet l'exécution de tâches en parallèle. Elle est utilisée pour : Empêcher le blocage de l'interface graphique durant l'exécution d'un programme. Actualiser les messages affichés dans l'interface graphique.
- **Sys** : Facilite les interactions avec l'interpréteur Python, comme le passage de l'argument du port lors du lancement du serveur.
- **PyQt6** : Permet de créer une interface graphique conviviale, simplifiant les interactions utilisateur.
- **Os** : Permet d'interagir avec le système d'exploitation.

Compatibilité :

- Windows 10, Windows 11
- Linux Debian
- MacOS

Déploiement :

1) Prérequis :

- Installation de Python 3.10 ou plus récente (serveur et clients)
- Installer les bibliothèques Python nécessaire pour le bon fonctionnement du programme : Sur la machine client dans un interpréteur python lancer la commande : «pip install Pyqt6 »

(Les autres bibliothèques utilisées sont natives à Python et n'ont pas besoin d'installation supplémentaire.)

- Installer les interpréteurs nécessaires pour exécuter les programmes sur la machine serveur :

  GCC pour les programmes en C

  Node.js pour les programmes en JavaScript

2) Installation Serveur :

- Copier le fichier serveur.py dans un dossier dédié sur la machine serveur.
- Depuis un terminal, naviguer grâce à la commande cd jusqu'au dossier contenant le fichier serveur.py et exécuter la commande : « python serveur.py <port> » Windows et « python3 serveur.py <port> » Linux
  Remplacez <port> par le numéro de port choisi pour écouter les connexions

3) Installation client :

- Copier les fichiers client.py dans un dossier dédié sur la machine client.
- Dans le dossier ou est présent client.py crée dossier config, puis crée dans ce dossier un fichier config.txt

- Modifier le fichier config.txt pour y inscrire les adresses IP et les ports des serveurs disponibles (obtentions des ip des serveurs en faisant un ip config sur Windows ou un ip a sur linux), au format suivant : ip_serveur|port_du_serveur
  ip_serveur : ip du serveur port_du_serveur = port du serveur
  Exemple :
  192.168.1.1|1001
  192.168.1.2|1002
- Ensuite depuis un terminal, naviguer grâce à la commande cd jusqu'au dossier contenant le fichier client.py et exécuter la commande : « python client.py » pour Windows et « python3 client.py » Linux

Pour un déploiement plus détailler voir documentation d'installation

Tests :

Tests de connectivité :

Lancer le serveur et le client sur des machines distinctes :

Vérifier que le client peut se connecter au serveur en renseignant l'IP corrects dans l'interface graphique et que le bon port est présent dans la config → si dans l'interface graphique est écrit le message 'Envoie du Programme' le Client est bien connecter au serveur

D'autres messages d'erreur peuvent s'afficher si le serveur est déjà occupé ou si le serveur n'est pas joignable.

Tests d'exécution :

Depuis le client, envoyer un programme simple (exemple : un script simple qui affiche "Hello World").

Vérifier que le serveur exécute le programme et renvoie la sortie standard ou une erreur si le script a une erreur.

Tests de gestion avec plusieurs clients :

Simuler plusieurs connexions client simultanées en lançant plusieurs clients

Vérifier qu'un seul client peut exécuter un programme et que les autres sont rejeter. S'ils sont rejetés les ip inscrites dans config.txt devrait s'afficher pour indiquer les autres ip joignables

Défis rencontrés et solutions mise en œuvre

Nom	Défi	Solution
Envoi du programme au serveur	Le client envoyait les données trop rapidement sans attendre une réponse, causant des pertes de données.	Ajout d'une confirmation du serveur avant d'envoyer chaque ligne de code.
Connectivité client/interface	L'interface graphique se bloquait pendant l'exécution du programme faute de multi-threading.	Utilisation de `threading` pour exécuter les tâches en parallèle.
Accès concurrent au serveur	Les clients étaient bloqués ou recevaient des erreurs si un programme était en cours d'exécution.	Mise en place d'un verrou (lock) pour rediriger les clients vers une attente si le serveur était occupé.

Réflexion sur les améliorations possibles

File d'attente :

Implémenter une file d'attente pour les clients se connectant au serveur lorsque celui-ci est occupé. Chaque client serait notifié de sa position dans la file.

Base de données des logs :

Ajouter une base de données (par exemple SQL) pour enregistrer les connexions au serveur, les programmes exécutés et leurs résultats.

Améliorations de l'interface graphique :

Ajouter une barre de progression estimant le temps restant pour l'exécution du programme.