Vargiu Matteo RT21 ROM

Document de réponse SAE302 :

Table des matières :

Introduction :

Dans le cadre de la SAE 302 ce projet a pour objectif de mettre en place une architecture multi-serveurs capable de gérer les requêtes de clients pour compiler et exécuter des programmes dans différents langages, ici nous avons Python, Java, C, et C++. J'ai choisi de réaliser la version simplifiée du sujet, en me concentrant sur les fonctionnalités les plus importante pour respecter le temps et les contraintes.

L'objectif était donc d'assurer une communication entre un client et les serveurs tout en faisant une interface graphique simple et intuitive pour l'utilisateur. Ce projet m'a permis de mettre en pratique le réseau, la gestion de thread, mais aussi le développement d'interfaces graphiques avec PyQt5.

Réponse au cahier des charges :

Fonctionnalités :

Fonctionnalités Réalisées :

- **Connexion Client-Serveur :**
  Mise en place d'une communication réseau entre un client et serveur via des sockets.
  Gestion des connexions multiples avec un message pour informer les clients si un serveur est déjà occupé.

- **Compilation et Exécution de Programmes :**
  Prise en charge des programmes en Python, Java, C, et C++ avec détection du langage.
  Compilation pour les langages nécessitant un exécutable (Java, C, et C++).
  Récupération des résultats ou des messages d'erreur pour les envoyer au client.

- **Interface Graphique Client :**
  Développement d'une interface graphique avec PyQt5 pour faciliter l'utilisation du système.
  Possibilité de choisir un fichier à envoyer au serveur.
  Affichage du temps d'exécution, des résultats ou des erreurs reçus.

- **Gestion de l'Occupation des Serveurs :**
  Un serveur en cours d'exécution d'un programme refuse les connexions supplémentaires, invitant les clients à se connecter à un autre serveur disponible.

- **Timer Dynamique :**
  Ajout d'un compteur affichant le temps écoulé depuis l'envoi du programme au serveur jusqu'à ce que le résultat apparaisse.

- **Serveurs simultanés :**
  Possibilité de lancer plusieurs serveurs en simultané en argument en choisissant le port.

Fonctionnalités Non Réalisées :

- **Load Balancing :**
  Répartition intelligente des charges entre les serveurs actifs pour optimiser les performances.

- **Monitoring des Serveurs :**
  Interface ou système permettant de visualiser l'état des serveurs en temps réel.

- **Sécurité Avancée :**
  Ajout de mesures de sécurité comme le chiffrement des données échangées ou l'authentification client-serveur.

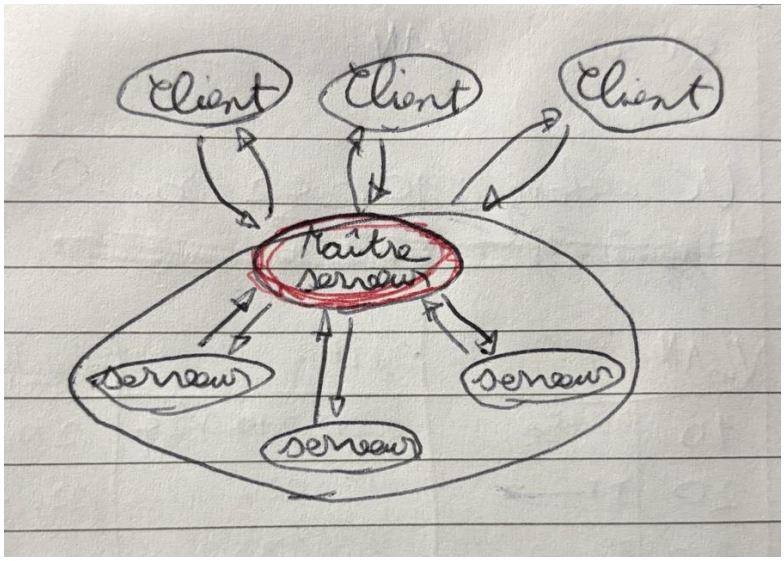
- **Améliorations de l'Interface Graphique :**
  Interface plus avancée avec des options supplémentaires comme un journal des résultats ou des configurations personnalisées.

- **Edition du programme :**
  Possibilité d'éditer/modifier le code directement sur l'interface graphique avant l'exécution.

Architecture de la solution :

Dans cette partie nous allons voir l'architecture du projet et de la solution avec une vue d'ensemble puis sur le projet simplifié en particulier.

Schéma de l'architecture complexes :



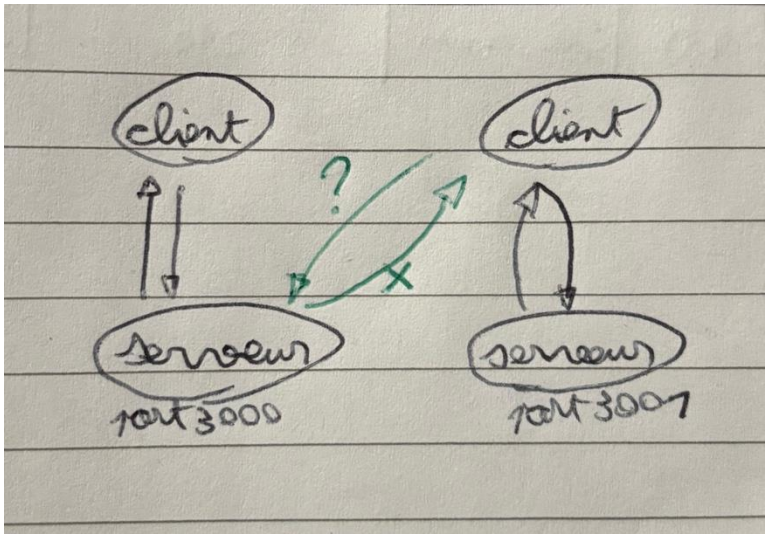
Description des éléments :

- **Client :**
  Se connecte et envoie un programme au maître serveur. Visualise les résultats et l'état des serveurs.

- **Maître serveur :**
  Répartit les requêtes des clients entre les serveurs en fonction de leur disponibilité et de leur charge.

- **Serveurs :**
  Traitent les requêtes, identifient le langage, compile ou exécute le code, puis renvoie les résultats ou signalent une surcharge au maître serveur.

Schéma de l'architecture simplifié :



Description des éléments :

- **Client :**
  Se connecte et envoie un programme à un serveur spécifique. Visualise les résultats et l'état du serveur.

- **Serveurs :**
  Reçoit le fichier du client, identifient le langage, compile ou exécute le code, puis renvoie les résultats. Si un serveur est occupé, il indique au client de se connecter à un autre comme ici en vert.

Architecture du code :

A présent, nous verrons comment le code du projet est structuré, ainsi que les bibliothèques utilisées. Cette vue d'ensemble permet de comprendre le fonctionnement global notamment en passant en revue les différentes fonctions.

Bibliothèques utilisées :

- ***socket*** : Pour la communication réseau entre client et serveur.
- ***threading*** : Pour permettre au serveur de gérer plusieurs clients
- ***os*** : Pour gérer les fichiers temporaires
- ***sys*** : Pour la gestion des arguments ou des erreurs système.
- ***PyQt5.QtWidgets*** et ***PyQt5.QtCore*** : pour l'interface utilisateur graphique et la gestion du timer.
- ***subprocess*** : Pour exécuter les programmes compilés ou interprétés.

Le Serveur :

Le fichier du serveur s'appelle ***serveurSAE302.py*** et repose sur plusieurs fonctions pour gérer les connexions client et exécuter les programmes envoyés.

Fonctions principales dans le serveur :

- __init__(Constructeur) **:**
  Permet d'initialiser l'adresse IP du serveur, les ports à écouter, et une variable *serveur_occupe* pour contrôler si le serveur est disponible ou non.

- __démarrer
  Crée un thread par port spécifié et démarre les serveurs.
  Permet au serveur de gérer plusieurs ports, même si une seule connexion est acceptée par port.
- __démarrer_serveur
  Fonction d'écoute, qui accepte ou refuse les connexions des clients en fonction de la disponibilité.
  Lance un thread dédié pour gérer le client avec *__gérer_client*.

- __gérer_client
  Fonction dédiée à la gestion d'un client.
  Récupère le programme envoyé par le client.
  Appelle la fonction appropriée pour exécuter le programme selon le langage détecté.
  Renvoie le résultat ou un message d'erreur au client.

- __exécuter_programme
  Détecte le langage du fichier reçu (C, C++, Python, Java) en fonction de son contenu, puis appelle la fonction d'exécution correspondante.

- __exécuter_ **(**python, c, cpp, java)
  Ces fonctions prennent en charge l'exécution des programmes dans différents langages :
  Créent des fichiers temporaires contenant le code.
  Pour C et C++, compilent les fichiers avec *gcc* ou *g++.*
  Pour Java, compilent avec *javac*.
  Pour Python, exécutent directement le script avec *python3*.
  Chaque fonction gère également les erreurs de compilation, d'exécution ou de nettoyage des fichiers temporaires.

Le Client :

Le fichier du client s'appelle ***clientSAE302.py***. Il utilise **PyQt5** pour une interface graphique conviviale et des sockets pour communiquer avec le serveur.

Fonctions principales dans le client :

- __choisir_fichier
  Ouvre une boîte de dialogue pour permettre à l'utilisateur de sélectionner un fichier.
  Vérifie que le fichier est bien sélectionné.
  Affiche son chemin dans l'interface utilisateur.

- __connexion_serveur
  Se connecte au serveur en fonction de l'IP et du port saisis.
  Envoie une requête au serveur.
  Affiche si la connexion est réussie ou si le serveur est occupé.

- __envoyer_programme
  Envoie le fichier sélectionné au serveur.
  Vérifie que le client est connecté et qu'un fichier a bien été sélectionné.
  Envoie le contenu du fichier au serveur pour exécution.

- __majcompteur_verifresult
  Gère le timer affiché dans l'interface utilisateur.
  Met à jour le timer toutes les 0,1 secondes jusqu'à réception du résultat du serveur.
  Stoppe le timer et affiche le résultat dans une zone de texte.

Planning :

Phase	Début	Fin	Durée(jours)	Description
Conception	12/11	27/11	16	Planification et réflexion sur le projet
Développement	27/11	27/12	31	Codage, mise en œuvre des fonctionnalités.
Livrables	28/12	31/12	4	Documentation et préparation du rendu.

- **Phase de Conception :**
  Définition des objectifs et compréhension du cahier des charges.
  Choix des technologies adaptées : utilisation de Python, PyQt5, et sockets réseau.
  Pour la répartition des tâches j'ai choisi d'avancer petit à petit des deux côtés serveur et client.

- **Phase de Développement :**
  Codage du serveur : gestion des connexions, exécution des programmes, et traitement des différents langages.
  Développement du client avec une interface graphique en PyQt5.

- **Phase de Finalisation et Documentation :**
  Création du document de réponse, de la vidéo de démonstration et du document d'installation.

Conclusion :

Aspects professionnels :

Ce projet m'a permis de développer des compétences techniques variées, notamment dans la gestion de sockets, la programmation en Python, et l'utilisation de PyQt5 pour créer une interface graphique.
Même si toutes les fonctionnalités demandées n'ont pas été réalisées, la structure mise en place respecte les objectifs du sujet simplifié.

Les points d'améliorations possibles sont :
- **La sécurité** : Absence de cryptage pour sécuriser les échanges entre client et serveur.
- **Le monitoring et scalabilité** : Pas de suivi dynamique des ressources des serveurs ni de mécanisme avancé de répartition des charges.
- **L'ergonomie de l'interface** : L'interface reste basique et pourrait être améliorée pour une meilleure expérience utilisateur.

Ce projet m'a aussi permis d'apprendre à gérer des contraintes, comme le respect du cahier des charges et l'utilisation de bibliothèques limitées.

Ressenti personnel :

D'un point de vue personnel, le projet a été un défi intéressant mais très dur à gérer.
Certaines étapes ont été plus complexes que prévu, notamment la gestion des threads et des sockets, qui m'ont demandé plusieurs recherches et essais avant d'arriver à une solution fonctionnelle. J'ai aussi eu du mal à gérer mon temps et à étaler le travail, j'ai plutôt fonctionné en grosse session de travail éparpiller dans le temps. J'ai aussi rencontré des difficultés lors du démarrage, j'avais en quelque sorte peur de me lancer car je pensais que je serais incapable de faire un tel projet. Mais finalement petit à petit j'y suis bien arrivé. Bien que le temps et l'organisation ait été une contrainte, j'ai beaucoup appris et apprécié ce projet, ce qui m'a permis de progresser en code et dans la gestion de projet.