SAE-302-Développer une application communicante

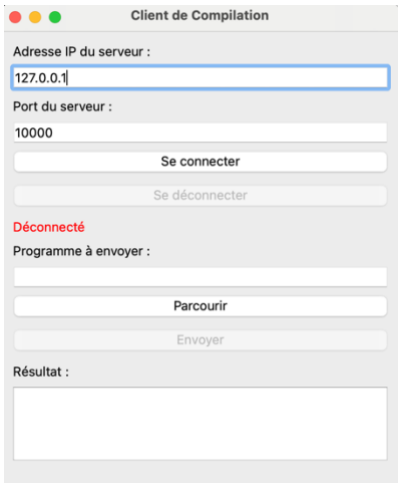
Introduction

Ce projet a pour objectif de développer un système client-serveur où chaque serveur ne peut accepter qu'un seul client à la fois. Si un second client tente de se connecter, il reçoit un message indiquant : "Le serveur est occupé, veuillez vous connecter à un autre serveur." Le serveur est capable de compiler et d'exécuter des programmes écrits dans divers langages tels que C, C++, Java et Python. Du côté client, une interface graphique simple permettra de se connecter à un serveur en spécifiant l'adresse IP et le port. Une fois la connexion établie, le client peut envoyer des programmes au serveur, et le serveur renverra les résultats ou les erreurs correspondants à la fin de l'exécution.

Fonctionnalités

Côté Client

1. **Connexion au serveur**
   - Le client peut se connecter à un serveur en entrant une adresse IP et un port, avec des valeurs par défaut disponibles.
   - Le statut de connexion est affiché clairement (connecté ou déconnecté).
   - Une option pour se déconnecter proprement est prévue.
2. **Envoi de programmes**
   - Le client peut sélectionner un fichier contenant le code à envoyer via une interface graphique.
   - Le programme est transmis au serveur pour compilation et exécution.
   - Les résultats (ou erreurs) sont affichés dans une zone de texte.
3. **Interface graphique (GUI)**
   - Utilisation de PyQt6 pour créer une interface graphique intuitive.
   - Présence de champs de saisie, boutons d'action, et visualisation des résultats.



Côté Serveur

1. **Gestion des connexions**
    - Acceptation de plusieurs connexions simultanées grâce à l'implémentation de threads.
    - Rejet des nouveaux clients lorsque le serveur est déjà occupé.
2. **Compilation et exécution**
    - Détection automatique du langage du programme envoyé (Python, C, C++, Java).
    - Compilation et exécution des programmes avec gestion des erreurs.
3. **Transmission des résultats**
    - Envoi des résultats ou des messages d'erreur au client.

Architecture de la solution

L'architecture de ce projet repose sur deux composants principaux : le client et le serveur. Ces deux parties communiquent entre elles via le protocole TCP, qui garantit une transmission fiable des données.

1. Technologies utilisées

- **Python** : Langage principal pour le développement.
- **PyQt6** : Framework utilisé pour créer l'interface graphique du client.
- **Sockets** : Mécanisme de communication réseau basé sur le protocole TCP.
- **Threads** : Gestion des connexions simultanées sur le serveur.

2. Structure du système

- **Client** : Le client est une application graphique qui permet à l'utilisateur de se connecter à un serveur en entrant une adresse IP et un port. Il peut ensuite envoyer des programmes à compiler et recevoir les résultats.
- **Serveur** : Le serveur reçoit les programmes envoyés par le client, détecte automatiquement leur langage, les compile, les exécute et transmet les résultats ou les messages d'erreur au client.
- **Processus:**
    1. Le client se connecte au serveur en spécifiant l'adresse IP et le port.
    2. Une fois connecté, le client peut choisir un fichier programme à envoyer.
    3. Le serveur compile et exécute le programme.
    4. Les résultats ou les erreurs sont renvoyés au client.

3. Communication Client-Serveur

La communication entre le client et le serveur repose sur un protocole TCP pour assurer une connexion stable et fiable. Le client envoie des messages et des fichiers au serveur, qui répond avec les résultats de l'exécution. La gestion des connexions simultanées est effectuée par des threads sur le serveur, garantissant qu'une seule connexion est active à la fois.

Cette architecture est adaptée à un environnement où la priorité est mise sur la stabilité et la simplicité. Elle fournit une base solide pour intégrer des améliorations comme la gestion de plusieurs clients ou l'ajout de nouveaux langages à l'avenir.

Planning du projet

Pour réaliser ce projet, j'ai suivi un planning structuré :

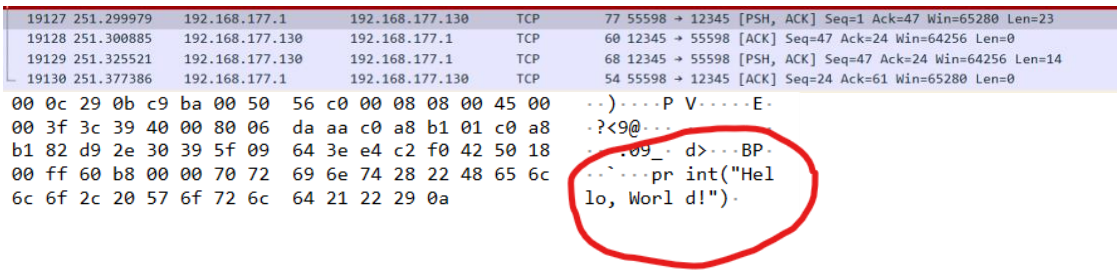
1. **Du 15 novembre au 29 novembre** : J'ai travaillé sur le serveur. Cela incluait la gestion des connexions, la détection des langages de programmation, ainsi que la compilation et l'exécution des programmes envoyés par les clients.
2. **Du 1er décembre au 20 décembre** : J'ai développé le client et l'interface graphique. Cette partie a permis de concevoir un outil permettant de se connecter au serveur, d'envoyer des programmes et de visualiser les résultats d'exécution.
3. **Du 20 décembre au 31 décembre** : J'ai rédigé le document d'installation et le rapport final, et j'ai réalisé une vidéo de présentation. Pendant cette période, j'ai aussi effectué des tests approfondis pour vérifier que tout fonctionnait correctement.

Défis rencontrés et solutions

1. **Détection du langage de programmation**
    - **Problème** : Identifier le langage du programme envoyé.
    - **Solution** : Analyse des mots-clés spécifiques au langage (e.g., "#include" pour C/C++, "public class" pour Java).
2. **Surcharge du serveur**
    - **Problème** : Gérer les connexions lorsque le serveur est occupé.
    - **Solution** : Notifier le client et lui permettre de choisir un autre serveur.

Améliorations possibles

- Ajouter une file d'attente pour les connexions lorsque le serveur est occupé.
- Intégrer un chiffrement pour sécuriser les échanges entre le client et le serveur.



En l'état actuel, il est facile d'observer les échanges entre le client et le serveur, comme le montre l'exemple capturé avec Wireshark. Les captures d'écran ci-jointes montrent que les communications, y compris les requêtes du client et les réponses du serveur, sont transmises en texte clair.

- Améliorer l’interface graphique pour une meilleure expérience utilisateur.

Conclusion

Le projet met en place une solution client-serveur performante et flexible pour compiler et exécuter des programmes. L'utilisation de threads et d'outils modernes tels que PyQt6 ou PyQt5 garantit de bonnes performances et une expérience utilisateur fluide. Les défis rencontrés ont permis d'acquérir des compétences et de résoudre des problèmes liés aux systèmes distribués. Des améliorations futures pourraient encore améliorer la portabilité, la sécurité et la robustesse du système.

Hamed AL-ABRI