Rapport Final du Projet

Introduction

Ce projet visait à développer une solution client-serveur permettant l'exécution de programmes écrits dans plusieurs langages de programmation (Python, C, et C++). L'objectif était de créer un système fonctionnel, robuste et capable de gérer plusieurs clients. Ce document présente les détails techniques du projet, les choix technologiques, les problèmes rencontrés, et les solutions mises en place.

Le projet inclut :

- Une **interface graphique conviviale pour le client**.
- Un **serveur multithread** capable de compiler et exécuter des fichiers envoyés par les clients.
- La prise en charge de plusieurs langages de programmation avec des mécanismes spécifiques de compilation et d'exécution.

Le Client

1. Description du client

Le client est une application développée en **Python** avec la bibliothèque graphique **PyQt6**. Il offre les fonctionnalités suivantes :

➔ **Connexion au serveur** : Permet de se connecter à un serveur distant.
➔ **Sélection de fichiers** : Accepte les fichiers .py, .c, et .cpp.
➔ **Envoi au serveur** : Transmet les fichiers pour exécution.

-> Affichage des résultats : Affiche les résultats retournés par le serveur, y compris le temps d'exécution.

2. Fonctionnalités principales

-> Connexion au serveur :
Le client utilise des sockets pour établir une connexion avec le serveur. Une fois connecté, il est prêt à envoyer des fichiers.

-> Interface utilisateur :
L'interface est intuitive et comprend des boutons pour se connecter, sélectionner un fichier, et envoyer le fichier au serveur.

3. Exemple de code clé

Voici un exemple de la connexion au serveur depuis le client :

```python
def se_connecter_au_serveur(self):
    # Connexion au serveur
    try:
        self.socket_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket_client.connect((self.hote_serveur, self.port_serveur))
        self.affichage_resultat.append("Connecté au serveur.")  # Afficher un message si connecté
        self.bouton_selectionner_fichier.setEnabled(True)  # Activer le bouton pour sélectionner un fichier
        self.bouton_envoyer.setEnabled(True)  # Activer le bouton pour envoyer le fichier
    except Exception as e:
        self.affichage_resultat.append(f"Erreur de connexion : {e}")  # Message en cas d'erreur
```

Connexion au Serveur

Le serveur et le client communiquent à l'aide de **sockets**. La connexion se déroule en trois étapes principales :

1. Le serveur est démarré et attend des connexions sur un port spécifique.
2. Le client se connecte au serveur en utilisant son adresse IP et le port configuré.
3. Une fois connecté, une communication bidirectionnelle est établie pour l'envoi de fichiers et la réception des résultats.

Envoi des Fichiers

1. Processus

Le client permet à l'utilisateur de sélectionner un fichier (Python, C, ou C++) via un explorateur de fichiers intégré. Une fois sélectionné, le fichier est envoyé au serveur.

2. Gestion des fichiers

Le type de fichier est déterminé par son extension (.py, .c, .cpp). Le serveur applique ensuite la méthode de traitement appropriée.

Exemple de code pour l'envoi des fichiers :

```python
def selectionner_fichier(self):
    # Sélectionner un fichier Python, C ou C++
    chemin_fichier, _ = QFileDialog.getOpenFileName(
        self,
        "Sélectionner un fichier",
        "",
        "Python Files (*.py);;C Files (*.c);;C++ Files (*.cpp)"
    )
    if chemin_fichier:
        self.fichier_selectionne = chemin_fichier
        self.etiquette_fichier.setText(f"Fichier sélectionné : {chemin_fichier}")  # Afficher le chemin
    else:
        self.etiquette_fichier.setText("Aucun fichier sélectionné.")  # Message si aucun fichier n'est choisi
```

Le Serveur

1. Description

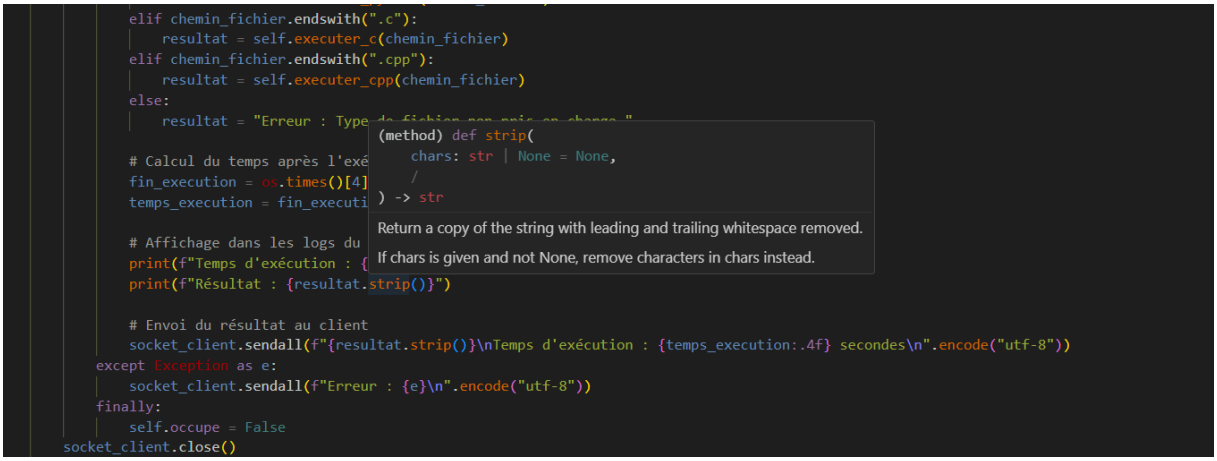
Le serveur est un programme capable de gérer plusieurs clients en parallèle. Il utilise des threads pour éviter le blocage des requêtes. Chaque client est traité indépendamment.

2. Fonctionnalités principales

- **Gestion des connexions multiples :** Utilisation de threads pour traiter chaque client individuellement.
- **Traitement des fichiers :** Détection du type de fichier et application de la méthode de compilation/exécution appropriée.
- **Logs détaillés :** Le serveur enregistre les connexions des clients, les fichiers reçus, et les résultats.

3. Exemple de code clé

Voici un extrait montrant la gestion des clients sur le serveur :



Compilation des Langages de Programmation

1. Python

-> Le fichier est exécuté directement à l'aide de l'interpréteur Python.

-> **Exemple de commande :**

```
execution = subprocess.run(["python", chemin_fichier], capture_output=True, text=True)
```

2. C

-> Le fichier est d'abord compilé avec GCC, puis exécuté.

-> **Exemple de commande :**

```
execution = subprocess.run([fichier_exe], capture_output=True, text=True)
```

3. C++

-> Le fichier est compilé avec G++, puis exécuté.

-> **Exemple de commande :**

```
compilation = subprocess.run(["g++", chemin_fichier, "-o", fichier_exe], capture_output=True, text=True)
```

Technologies Utilisées

- ➔ **Sockets :** Pour établir une connexion réseau entre le client et le serveur.
- ➔ **Threads :** Pour gérer plusieurs clients simultanément sans bloquer les autres.
- ➔ **PyQt6 :** Pour développer l'interface graphique du client.
- ➔ **Subprocess :** Pour exécuter les commandes de compilation et d'exécution.

Problèmes Rencontrés et Solutions

1. **Problème de compilation avec GCC/G++**

   ➔ J'ai eu plusieurs soucis pour la compilation avec des erreurs comme quoi le type de fichier n'était pas pris en charge. Enfaite le soucis venais que j'utilise mal le subprocess

2. **Problème de type de fichier**

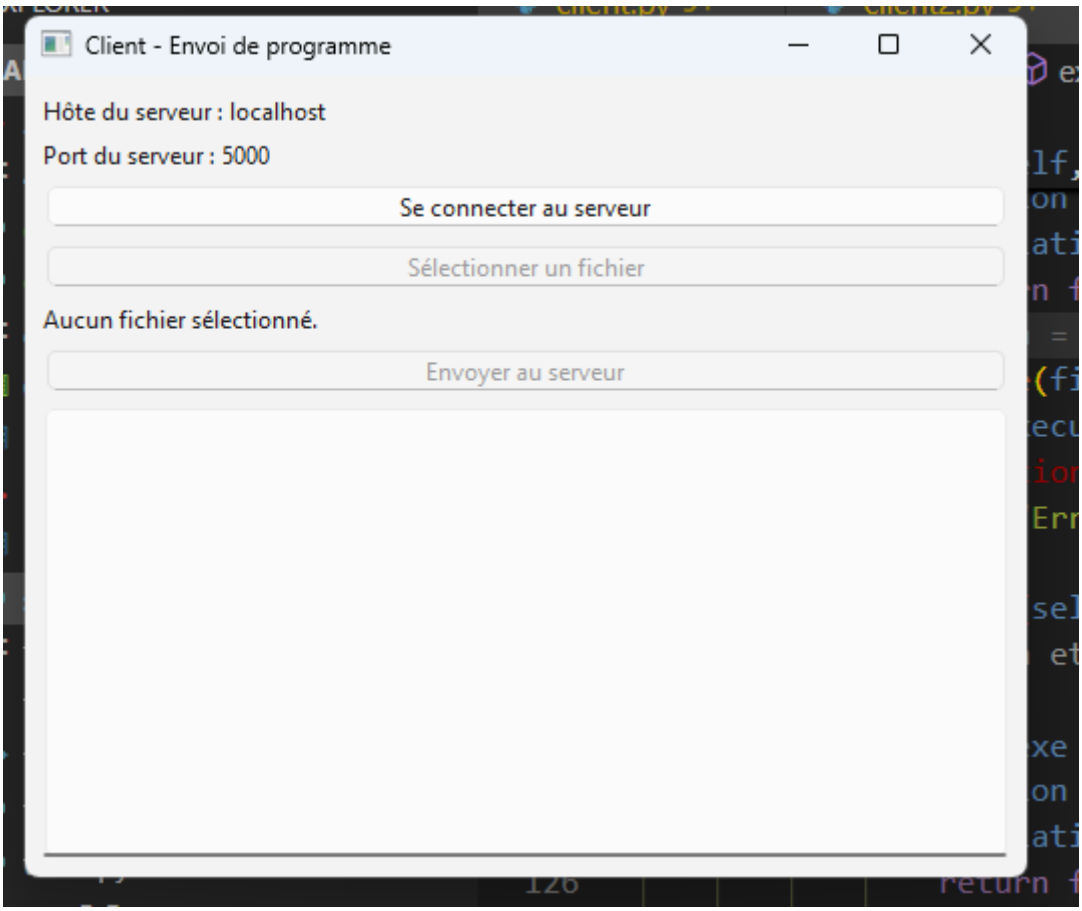
   ➔ Certains fichiers mal nommés étaient mal interprétés. Parce que l'extension était mal gérer par le code.

3. **Problème de maintien de connexion**

   ➔ Le client se déconnectait après chaque exécution. J'ai du modifier la gestion des threads pour maintenir la connexion.

Captures d'Écran

Exemple de l'interface graphique du client



Logs du serveur

```
PS C:\Users\yanis\OneDrive\Bureau\IUT Colmar\SAE 3.02> python server.py
Serveur démarré sur localhost:5000
Connexion reçue de : ('127.0.0.1', 50139)
Fichier reçu : C:/Users/yanis/OneDrive/Bureau/IUT Colmar/SAE 3.02/test.py
Temps d'exécution : 0.0000 secondes
Résultat : Hello World
```

Conclusion

Le projet a permis de mettre en pratique plusieurs concepts clés en informatique :

- ➔ Communication réseau via sockets.
- ➔ Gestion des threads pour la concurrence.
- ➔ Compilation et exécution de fichiers en Python, C, et C++.

Des améliorations potentielles incluent :

- ➔ Ajouter plus de langages pris en charge.
- ➔ Améliorer les logs et la gestion des erreurs.

Vidéo de démonstration :

https://youtu.be/jfan0FmuZio