Document de réponse - SAE302 Développer un cluster de calculs (Sujet Simple)

Table des matières

Introduction

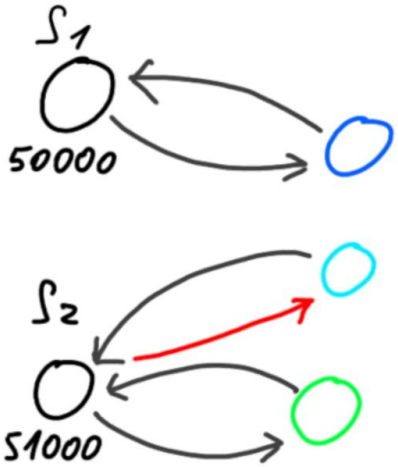
L'objectif de cette SAE était de concevoir et de développer une architecture multi-serveurs capable de recevoir des requêtes de clients, de compiler et d'exécuter des programmes dans un langage de programmation défini... Cela dit, étant en alternance et ayant beaucoup moins de temps au sein de l'IUT, un sujet plus simple nous a été proposé.

Ce sujet simple nous demande donc de coder un serveur pouvant recevoir et gérer des requêtes de plusieurs clients, d'exécuter les codes des clients s'il n'est pas occupé et de retourner une erreur s'il est déjà entrain d'exécuter un programme. Une fois le programme exécuté, le serveur le revoie au client et se rend à nouveau disponible pour d'autres.

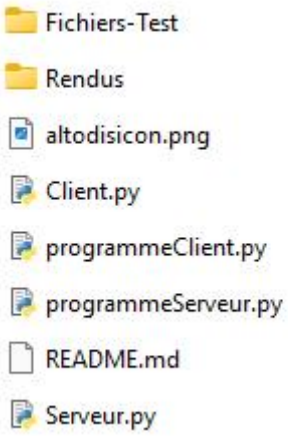
Le port du serveur doit pouvoir être précisé en argument lorsqu'on le lance et le client doit pouvoir le choisir graphiquement. Depuis son interface graphique, le client peut : se connecter à un serveur en précisant son ip et son port, envoyer un fichier de code et attend le résultat sans bloquer avec un chrono qui affiche le temps d'attente.

Architecture du projet

Le projet version simple est donc composé de deux choses : des clients et des serveurs. Chaque serveur est indépendant de l'autre mais il peuvent tourner en même temps, tout comme les clients. Sur le schéma, à droite en couleurs, les clients qui peuvent se connecter à plusieurs sur un serveur mais uniquement s'il n'est pas en cours d'exécution de code, sinon il refuse comme en dessous.



Voici les fichiers présents dans le dossier du projet :



Les serveurs sont codés de la façon suivante : un fichier de classe «Serveur.py» qui définit la classe et toutes ses méthodes ainsi qu'un fichier «programmeServeur.py» qui fait appel à cette classe et qui crée un serveur en fonction des arguments de la commande pour le lancer et gère les erreurs dans une boucle qui ne s'arrête que si le serveur en reçoit la demande.

Pour les clients, c'est un peu le même paterne, un fichier de classe «Client.py» où ils sont définis avec leur partie graphique et un fichier «programmeClient.py» qui crée l'affichage avec le client.

L'image sert uniquement d'icon pour la partie graphique des clients et le dossier «Fichier-Test» comporte des fichiers de chaque langage qui peuvent servir à tester le fonctionnement du code.

Librairies utilisées

Librairies natives

Les librairies natives utilisées dans ce projet sont :

- **socket** : afin de créer des connexions réseau entre les clients et les serveurs grâce à des adresses ip et des ports.
- **time** : permet d'ajouter une attente avec time.sleep pour simuler une congestion sur un serveur et pour créer un chronometre chez le client afin de voir le temps d'exécution.
- **threading** : pour éxécuter différentes tâches en même temps tel que l'écoute de requête, la gestion de connexion clients et l'envoie de réponses.
- **os** : afin d'intéragir avec les fichiers et leurs noms et chemins dans le système.
- **subprocess** : pour faire appel à des commandes du systèmes pour lire, compiler et exécuter les codes, avec subprocess.run et les options qui permettent de faire ce que l'on veut dans le système.

Librairies à télécharger

- **PyQt6** : PyQt6 est une librairie qui permet de créer des affichages graphiques assez facilement en utilisant des Layout et des Widgets. L'interface graphique a été utiliser du côté du client pour utiliser toutes ses fonctionnalités.

Fonctionnalités implémentées

Serveur

- **Lancement du serveur sur un port donné en argument** :
- **Gestion de multiples connexions de clients** : grâce à une méthode de classe «connexion»; le serveur peut accepter la connexion de plusieurs clients à la fois. Pour chacun d'entre eux, il lance un Thread de gestion.
- **Gestion des clients** : la méthode de gestion de client permet d'écouter les potentiels requêtes qu'un client peut faire et en fonction de cela, lui répondre ou lancer la méthode de gestion du fichier reçu.
- **Gestion des fichiers Clients** : le serveur est capable de gérer différents langages de programmation (à condition que les compilateurs/langages soient installés sur sa machine) ou bien de retourner une erreur si ce dernier n'est pas reconnu. Il fait appel à différentes méthodes en fonction du langage de programmation du fichier reçu. Voici les langages supportés :
  - **Python** : Lorsqu'il reçoit du code Python, le serveur utilise la librairie native **Python subprocess** avec sa fonction **run** et l'option «python» pour lancer le fichier avec

l'interpreteur Python. Il retourne et capture le résultat du code. En cas d'erreur, il renvoit une erreur dans le resultat.stderr et le restultat normal se situe dans le resultat.stdout.

- **C** : Pour exécuter du C, le serveur doit d'abord compiler le code et pour cela il fait appel au compilateur du system : **gcc**. La méthode de code C crée donc un fichier temporaire (qui est supprimé automatiquement à la fin) où il écrit le contenu du code. Ensuite, il fait aussi appel à subprocess.run mais cette fois ci avec les options «gcc» pour faire appel au compilateur, le fichier temporaire à compiler, «-o» pour préciser l'output donc la sortie et enfin le nom de l'exécutable. Gcc doit être installé sur la machine, si ce n'est pas le cas, il y aura une erreur lors de la création du fichier et donc le serveur renvoie uune erreur de compilation. Il réutilise ensuite subprocess.run pour lancer ce code et capture les sorties.
- **C++** : Pour C++, c'est exactement la même chose que le C sauf que le compilateur s'appelle g++.
- **Java** : Pour java, il est aussi question de créer un fichier temporaire mais celui doit avoir le même nom que les classes que l'on utilise dans le fichier donc il prend le nom du ficher que l'on donne. Le serveur uutilise ensuite subprocess avec javac pour compiler puis java pour lire et prendre la sortie.
- **Fichiers .txt** : il n'est pas vraiment question de code mais le serveur gère aussi les fichiers .txt, il revoie alors juste le contenu du fichier.

- **Envoie du résultat** : Le serveur a une méthode envoie qui est très simple et qui prend simplement le résultat du code ainsi que la connexion et envoie le résultat.
- **Fermeture de la connexion client** : le serveur ferme simplement sa connexion avec un client.
- **Arrêt du serveu**r : quand il reçoit une demande d'arrêt d'un client via la gestion, le serveur s'arrête proprement.
- **Gestion des potentielles erreurs** :

Client

- **Interface graphique** : Chaque client à sa propre interface graphique réalisée avec la librairie PyQt6 qui permet de gérer toutes ses fonctionnalités et ses méthodes que voici :
  - **Connexion à un serveur** : le client peut se connecter à un serveur grâce à son port et son ip qu'il peut préciser graphiquement dans un QLineEdit.
  - **Charger un fichier** : Avec un bouton «charger», un client peut séléctionner un fichier via un QFileDialog et ce dernier se verra affiché sur un QLineEdit.
  - **Modifier un fichier** : grâce au QLineEdit, le contenu du fichier sélectionné par le client pourra être modifié.
  - **Envoyer un fichier** : une fois le fichier sélectionné et potentiellement modifié, il peut être envoyé au serveur et le client attendra ensuite la réponse.
  - **Gestion du temps d'attente** : comme demandé dans le sujet, un client ne doit pas bloquer lorsqu'il attend. Il y a donc une gestion de cette attente avec un chronomètre qui se lance grâce à l'action attendreResultat. Cette action lance un thread tempsAttente qui va mettre à jour en direct grâce à QMetaObject.invokeMethod différents champs et surtout le temps. invokeMethod permet d'agir sur une interface

graphique depuis un thread qui n'est pas le même que cett interface. Ici, on utilise donc la méthode PyQt setText sur l'interface graphique pour modifier l'affichage du temps.

- **Déconnexion du serveur** : le client peut se déconnecter d'un serveur afin de se connecter à un autre serveur disponnible par exemple ou bien se reconnecter au même serveur plus tard.
- **Éteindre le serveur** : cela envoie au serveur un requête « arrêt» qu'il gère pour lui-même appeler sa fonctionnalité d'arrêt et s'éteindre.
- **Quitter la partie graphique du client** : avec QApplication.exit(0), on met simplement fin au programme du client.

Fonctionnalités manquantes

Voici les fonctionnalités qui auraient pu être ajoutées pour que le projet soit complet.

Serveur

Client

Problématiques

Professionnelles

Comme dit précédemment, les groupes étant en alternance avaient beaucoup moins de temps de cours prévus pour réaliser cette SAE car nous avions une période d'un mois en entreprise qui est arrivé pendant le projet. Cela a réduit le temps de travail que je pouvais effectuer sur le projet mais heureusement, après en avoir parlé avec mon maître d'apprentissage, j'ai pu travailler plusieurs heures sur mon projet en entreprise. Sans cela, je suis certain que mon projet aurait été beaucoup moins avancé.

Personnelles

La programmation n'est en soit pas quelque chose qui me dégoûte mais je n'excelle pas dans le domaine. Ce projet m'a largement poussé au-delà de mon niveau de base en programmation orienté objet, j'ai eu pas mal de problèmes et d'incompréhension durant la réalisation du projet comme la gestion des threads et de l'interface graphique pour mettre le temps à jour. Mais en passant beaucoup de temps dessus, j'ai pu comprendre et m'améliorer sur ces points.

Point sécurité

L'accès aux serveurs ne demande aucun mot de passe, ou aucune authentification ce qui peut poser problème dans le cas d'un logiciel qui peut être mis en place dans un réseau publique. Il suffit de connaître l'adresse ip de la machine ainsi que son port et on peut lui faire exécuter ce que l'on veut. De plus, la communication entre les clients et le serveur n'est pas cryptée. De ce fait,

Téléchargement et mise en place

Pour télécharger le projet, je vous redirige vers le document de procédure d'installation qui détail l'installation et le lancement des différents éléments du projet.

Vidéo de présentation

Vous pouvez visionner une démonstration du projet sur YouTube via le lien suivant :
https://www.youtube.com/watch?v=wImmwRUS4t4