Rapport Final:

SAÉ3.02 : Conception d'une Architecture Multi-Serveurs pour la Compilation et l'Exécution de Programmes

Table des matières

1.Introduction et Contexte

L'objectif de ce projet est de concevoir et de développer une architecture multi-serveurs permettant la compilation et l'exécution de programmes écrits dans différents langages (Python, C, C++, Java). L'architecture doit gérer plusieurs clients simultanément, répartir la charge entre un serveur maître et un ou plusieurs serveurs esclaves, et fournir une interface graphique côté client pour faciliter l'édition et l'envoi du code.

1.1. Objectifs Principaux

Recevoir du code (via un client graphique PyQt6) pour exécution.

Compiler et/ou interpréter le code (Python, C, C++, Java).

Gérer plusieurs clients simultanés (threading, sockets).

Répartir la charge : ne pas saturer le serveur maître au-delà d'un certain nombre de tâches (MAX_TASKS).

Déléguer au besoin des exécutions à des serveurs esclaves.

Fournir un mécanisme d'administration (consulter l'état du serveur, modifier la limite de tâches, etc.).

2. Architecture Globale

2.1. Présentation des Composants

L'architecture repose sur trois éléments :

2.1.1. Client

Pour l'interface du client j'ai utilisé PyQt6.

Sur l'interface du client on permet à l'utilisateur de renseigner l'IP/Port du serveur maître, de choisir un langage de programmation (Python, C, C++, Java), de créer ou importer un code source, puis de l'exécuter.

Le client affiche la sortie ou les erreurs de compilation on peut aussi vide la sortie si besoin

On peut envoyer des commandes ADMIN (GET_INFO, SET_MAX_TASKS et SET_MAX_SLAVES) au serveur maître pour gérer les paramètres du serveur du client

Il y a aussi un bouton de statuts pour afficher l'état de connexion avec le server rouge = non connecter vert = connecter.

2.1.2. Serveur Maître

Le serveur maître constitue le cœur de l'architecture. Il agit comme un point central pour gérer les requêtes des clients, distribuer les tâches aux serveurs esclaves, et administrer l'ensemble du système. Voici ses principales fonctionnalités :

Gestion des connexions client : Le serveur maître est capable de gérer plusieurs connexions simultanées en utilisant le threading. Chaque client qui se connecte au serveur est traité comme un processus indépendant, ce qui permet une gestion efficace des ressources.

Répartition de la charge : Le serveur maître surveille en permanence le nombre de tâches en cours (défini par MAX_TASKS). Lorsqu'il atteint ce seuil, il délègue automatiquement les nouvelles tâches à des serveurs esclaves. Cela garantit que le serveur maître ne soit pas surchargé.

Administration dynamique : À travers des commandes spéciales envoyées par le client, l'utilisateur peut :

Consulter l'état actuel du serveur (nombre de tâches, esclaves actifs, etc.).

Modifier la limite de tâches (MAX_TASKS) et celle de (MAX_SLAVES).

Lancement dynamique des esclaves : Le serveur maître est capable de démarrer de nouveaux serveurs esclaves lorsque la charge augmente. Il utilise des ports prédéfinis pour s'assurer que les esclaves se lancent sans conflit.

2.1.3. Serveurs Esclaves

Les serveurs esclaves sont des extensions du serveur maître. Ils prennent en charge les tâches déléguées, exécutent le code reçu, et renvoient les résultats au maître. Voici leurs caractéristiques principales :

Exécution indépendante : Une fois qu'un esclave reçoit une tâche, il la traite de manière autonome. Le serveur maître reste libre pour traiter d'autres requêtes.

Compatibilité multi-langages : Les esclaves prennent en charge les mêmes langages que le serveur maître (Python, C, C++, Java), pour la compilation et l'exécution.

Robustesse : Si un esclave ne répond pas ou échoue, le serveur maître peut rediriger la tâche vers un autre esclave ou la reprendre en local.

Arrêt dynamique : Lorsque la charge baisse en dessous d'un seuil défini, les esclaves inutilisés peuvent être arrêtés automatiquement pour économiser des ressources.

3. Fonctionnalités Clés

3.1. Gestion des Programmes

Le système prend en charge des langages variés, notamment Python, C, C++, et Java. Chaque programme est compilé ou interprété selon les besoins. Par exemple :

Python est exécuté directement via l'interpréteur.

Les fichiers C/C++ sont compilés avec GCC/G++ avant exécution.

Les programmes Java sont compilés avec javac puis exécutés avec la machine virtuelle Java.

Le client peut importer un fichier existant ou écrire du code directement dans l'interface graphique avant de l'envoyer au serveur.

3.2. Répartition Dynamique des Charges

Le serveur maître surveille en continu la charge actuelle. Lorsque MAX_TASKS est atteint, il lance un serveur esclave ou délègue la tâche à un esclave existant.

Les esclaves sont gérés dynamiquement : ils sont lancés lorsque nécessaire et arrêtés lorsque la charge diminue.

3.3. Commandes ADMIN

Les commandes ADMIN permettent une gestion flexible des serveurs :

GET_INFO : Permet de consulter l'état actuel du système, y compris le nombre de tâches en cours et le nombre d'esclaves actifs.

SET_MAX_TASKS : Modifie la limite de tâches que le serveur maître peut traiter en local.

SET_MAX_SLAVES : Ajuste dynamiquement le nombre maximum d'esclaves pouvant être lancés.

3.4. Interface Utilisateur

Le client que j'ai créé à une interface graphique intuitive et simple qui permet :

D'éditer, importer, et sauvegarder du code.

De sélectionner le langage de programmation.

De surveiller l'état de connexion au serveur (statut rouge/vert).

D'exécuter du code et de consulter les résultats ou les erreurs.

3.5 Technologies utilisées

Pour répondre aux besoins techniques et aux objectifs du projet, plusieurs technologies ont été choisies :

Python :

Utilisé comme langage principal pour son expressivité et sa capacité à gérer des tâches réseau, de threading, et d'interfaçage graphique. Python est particulièrement adapté pour construire des systèmes distribués grâce à sa vaste bibliothèque standard.

PyQt6 :

Cette bibliothèque a été utilisée pour développer l'interface graphique du client. Elle offre une grande flexibilité et permet de créer des interfaces modernes et intuitives. PyQt6 a permis d'ajouter des fonctionnalités avancées telles que la coloration syntaxique, l'édition de code en temps réel, et des boutons interactifs pour gérer les paramètres.

Sockets TCP :

Les sockets ont été utilisés pour la communication entre le client, le serveur maître, et les esclaves. Ce choix garantit une communication fiable et bidirectionnelle, essentielle pour gérer les requêtes et transmettre les résultats.

GCC/G++ :

Ces outils de compilation sont indispensables pour exécuter les programmes en C et C++. Ils ont été intégrés au système à travers des appels via la bibliothèque subprocess de Python.

TDM-GCC :

Une version spécifique de GCC adaptée à Windows, utilisée pour la compilation sur cette plateforme. TDM-GCC offre une compatibilité accrue avec l'environnement Windows tout en étant léger et facile à configurer.

Java et Javac :

Les programmes Java sont compilés avec javac et exécutés avec la JVM (Java Virtual Machine). L'intégration de ces outils a permis de supporter ce langage très répandu dans l'industrie.

OS et sys :

Ces modules Python ont été utilisés pour gérer les fichiers locaux, configurer les chemins d'accès, et adapter le comportement du système aux différentes plateformes (Windows/Linux).

4. Défis Rencontrés et Solutions

4.1. Gestion des Ports pour les Esclaves

Problème identifié : Lors du lancement de nouveaux esclaves, des ports étaient parfois occupés, empêchant leur démarrage correct.

Solution mise en œuvre : Une détection dynamique des ports libres a été implémentée. Si un port est déjà utilisé, le système passe automatiquement au suivant dans une plage prédéfinie.

4.2. Problèmes de Compilation en C/C++

Problème identifié : Lors de la compilation de fichiers C/C++, des erreurs comme \u00ab Permission denied \u00bb se produisaient à cause de fichiers verrouillés par des exécutions précédentes.

Solution mise en œuvre : Avant chaque compilation, les fichiers de sortie existants (comme test.exe) sont supprimés pour éviter les conflits. Cela garantit que chaque exécution commence dans un environnement propre.

4.3. Robustesse de la Communication

Problème identifié : Les connexions réseau pouvaient être interrompues, entraînant des erreurs ou des tâches perdues.\n- Solution mise en œuvre : Un système de reconnexion automatique a été ajouté. En cas d’échec temporaire, le client tente plusieurs fois de se reconnecter avant de signaler une erreur à l’utilisateur.

5. Réflexion sur les Améliorations Possibles

5.1. Persistance des Données

Actuellement, les tâches en cours ne sont pas sauvegardées en cas de panne ou de redémarrage du système. Une solution consisterait à intégrer une base de données ou un fichier de journalisation pour stocker les tâches et permettre leur reprise.

5.2 Sécurité

Les communications entre les clients et le serveur maître ne sont pas chiffrés. Ajouter un chiffrement SSL/TLS garantirait la confidentialité et l'intégrité des échanges, particulièrement dans des environnements sensibles.

5.3 Monitoring en Temps Réel

Une interface graphique de monitoring pourrait être ajoutée pour afficher l'état des esclaves (charge CPU, état des ports, tâches en cours). Cela faciliterait la gestion des ressources et la détection des problèmes.

5.4 Scalabilité

Bien que le système soit conçu pour plusieurs esclaves, il pourrait être optimisé pour prendre en charge des centaines d'esclaves avec des mécanismes de clustering avancés.

6. Conclusion

Ce projet m'a permis d'explorer et de résoudre des problématiques complexes liées à la gestion des processus distribués et à la communication réseau. Bien que le système actuel atteigne les objectifs fixés, il offre de nombreuses perspectives d'améliorations pour le rendre encore plus robuste, sécurisé et évolutif. J'ai beaucoup apprécié cette SAE cella ma permis d'apprendre encore plus sur le fonctionne de python et j'ai beaucoup aime le projet cela a été amusant à réaliser il m'a donné 1 à 2 idées pour des projet personnelles.

Pour le temps mis sur le projet je dois être environ à 54h-58h en classe et temps personnelle compris

Je suis content de ce que j'ai réalisé même si j'aurais bien voulu pousser les choses plus loin mais par peur de trop complexifier mon code et projet je me suis arrête à là par conséquent je pense essayer pendant mon temps libre de finir ce projet et ajoute des choses qui me sont venue à l'idée au cours du projet

Merci d'avoir pris la peine de me lire mon rapport je vous souhaite une agréable journée ou soirée