* Les librairies que vous pouvez utiliser sont : python, os, sys, socket, threading, toutes librairies natives, pyqt6, custum tk inter, subprocess
* Les langages que j'espère pouvoir utiliser : java, c++, c, python
* Il serait sympa de pouvoir voir/éditer le code sur le client
* Contexte L'objectif de ce projet est 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, tout en gérant dynamiquement la répartition des charges entre plusieurs serveurs pour garantir une exécution fluide et scalable. Cette architecture devra être conçue pour gérer eVicacement plusieurs clients simultanés et, en cas de surcharge d'un serveur, déléguer les tâches à d'autres serveurs du cluster. Description du projet Votre tâche sera de développer une application distribuée qui met en œuvre une architecture multi-serveurs capable de gérer les fonctionnalités suivantes : Connexion client-serveur • Le client doit pouvoir se connecter à un serveur et envoyer un programme écrit en Python. Le client sera développé sous format graphique permettant de spécifier l’IP du serveur (une valeur par défaut sera proposée) et un port (de même une valeur par défaut sera proposée). • Le serveur doit accepter le programme, le compiler (si nécessaire) et l'exécuter. • Le résultat de l'exécution (ou les erreurs de compilation) doit être renvoyé au client. Gestion de multiples clients Le serveur doit être capable de gérer simultanément plusieurs clients. Chaque client peut soumettre un programme, et le serveur doit pouvoir traiter plusieurs requêtes de façon concurrente. Load Balancing (Répartition de charge) • Si le serveur principal (ou maître) atteint une certaine limite en termes d’utilisation du CPU ou du nombre de programmes1 en file d'attente, il doit être capable de déléguer des tâches à un ou plusieurs serveurs secondaires (esclaves). • Les serveurs secondaires doivent pouvoir s'exécuter de manière autonome une fois lancés et renvoyer les résultats au serveur maître qui, à son tour, transmettra les résultats au client. • Le nombre maximum de programmes sera définit en paramètre du programme. 1 Ce nombre devra être un paramètre de lancement du programme 3 Mécanismes de communication • La communication entre le client et le serveur, ainsi qu’entre les serveurs, doit se faire via des sockets. • Il faudra également gérer la robustesse des connexions (gestion des erreurs, reconnexion en cas de défaillance, etc.). Contraintes techniques Compilation et Exécution des programmes clients : Vous devrez prévoir l’exécution de programmes dans un ou plusieurs langages (par exemple, un serveur qui peut compiler et exécuter des programmes Python ou C). Les outils de compilation et d'exécution (comme GCC pour C ou l’interpréteur Python) doivent être disponibles sur les serveurs. Utilisation de ressources système : Le projet doit intégrer des mécanismes permettant de surveiller la charge CPU et mémoire de chaque serveur. Ces informations seront utilisées pour décider de la répartition des charges entre les serveurs.

|  |  |  |
| --- | --- | --- |
| Client | Interface graphique |  |
|  | Intuitif | Facilité d'usage |
|  |  |  |
| Serveur | Load balancing | Y'a-il un vrai cluser ? Si je mets une limite de 3 processus, est ce que la répartion des processus sur d'autres serveurs est gérée ? |
|  | Est-il capable d'executer des codes python, java, C++, C ? |  |
|  | Gestion des erreurs (code ou absence d'interpreteur/compilateur) |  |
|  | Gestion de plusieurs clients |  |