Rapport technique

OPEN SOURCE

Brahim-Jonathan-Brice

Brahim Jonathan Brice

École :	Paris Ynov Campus
Date :	05 Avril 2024
Cours :	Open Source
Classe :	Mast 2 CyberSécurité

En développant les modes de communication entre le lanceur et ses clients, nous avons adopté plusieurs solutions pour garantir une interaction efficace et robuste. Voici un rapport décrivant ces solutions.

1. Manuel utilisateur

Sur une machine Linux, ouvrez un terminal et exécutez les commandes suivantes :

git clone https://github.com/Brissou1418/command_launcher

cd command_launcher

Cela téléchargera le projet depuis GitHub et vous placera dans le répertoire fraîchement cloné.

Dans le même terminal, exécutez la commande suivante pour compiler le projet : **make**

Cela exécutera le Makefile et générera les fichiers exécutables nécessaires pour utiliser le serveur et le client.

Utilisez un éditeur de texte pour ouvrir le fichier Demon.conf présent dans le répertoire cloné. Ce fichier contient des paramètres importants pour le fonctionnement du serveur. Voici ce que signifient ces paramètres :

- MIN_THREAD: Nombre minimum de threads que le serveur peut créer.
- MAX_THREAD: Nombre maximum de threads que le serveur peut créer.
- MAX_CONNECT_PER_THREAD: Nombre maximal de connexions que chaque thread peut gérer simultanéme nt.
- SHM_SIZE: Taille de la mémoire partagée utilisée pour la communication entre le serveur et les threads.

Voici un exemple de configuration possible pour Demon.conf :

- MIN_THREAD = 2
- MAX_THREAD = 10
- MAX_CONNECT_PER_THREAD = 5
- SHM_SIZE = 1024

Après avoir modifié les paramètres selon vos besoins, enregistrez le fichier Demon.conf.

Pour démarrer le serveur, exécutez la commande suivante dans le même terminal **: ./Serveur**

Pour exécuter un ou plusieurs clients, ouvrez un nouveau terminal et accédez au répertoire du projet. Ensuite, exécutez la commande suivante pour chaque client que vous souhaitez démarrer **: ./Client**

Suivez les instructions affichées par le programme client dans le terminal pour interagir avec lui. Vous pouvez exécuter plusieurs clients simultanément en ouvrant plusieurs terminaux et en lançant le client dans chacun d'eux.

Une fois que le client est lancé, suivez les instructions données par le programme pour envoyer des commandes au serveur et recevoir des réponses. Assurez-vous de lire attentivement les messages affichés à l'écran pour comprendre ce qui se passe.

Pour fermer un client, vous pouvez envoyer la commande "END" où il est en cours d'exécution ou utiliser le raccourci clavier Ctrl+C dans le terminal pour arrêter le processus du client.

En suivant ces étapes, vous pourrez utiliser le projet de lancement de commandes sur votre machine Linux. Assurez-vous de configurer correctement le fichier Demon.conf et de suivre les instructions données par les programmes pour une utilisation correcte.

2. Manuel Technique

En développant les modes de communication entre le lanceur et ses clients, nous avons adopté plusieurs solutions pour garantir une interaction efficace et robuste. Voici un manuel technique décrivant ces solutions :

Modes de Communication entre le Lanceur et ses Clients :

1. **Utilisation de Tubes Només :** Pour permettre la communication entre le lanceur et ses clients, nous avons opté pour l'utilisation de tubes nommés. Ces tubes permettent d'établir un

canal de communication bidirectionnel entre les processus, ce qui facilite l'échange de données.
2. **Création de Tubes Només Individuels :** Chaque client crée un tube nommé individuel pour recevoir les réponses du lanceur. Cela garantit que chaque client reçoit sa propre réponse sans interférence avec les autres clients.
3. **Utilisation de Mémoires Partagées :** Pour permettre une communication efficace entre le lanceur et ses clients, nous avons mis en place l'utilisation de mémoires partagées. Ces mémoires offrent un moyen de partager des données entre les processus de manière rapide et efficace.
4. **Utilisation de Sémaphores pour la Synchronisation :** Nous avons utilisé des sémaphores pour assurer une synchronisation correcte entre les processus. Les sémaphores permettent de contrôler l'accès aux ressources partagées et d'éviter les problèmes de concurrence.

Étapes dans le Programme d'un Serveur :

1. **Lecture de la Configuration :** Le serveur commence par lire les paramètres de configuration à partir d'un fichier de configuration spécifié. Ces paramètres incluent le nombre minimum et maximum de threads, la taille de la mémoire partagée, etc.
2. **Gestion des Signaux :** Le serveur gère les signaux tels que SIGINT pour permettre une terminaison propre du programme en cas d'interruption.
3. **Création des Threads de Traitement :** Le serveur crée un certain nombre de threads pour traiter les demandes des clients. Ces threads sont créés dynamiquement en fonction des paramètres de configuration.
4. **Communication avec les Clients :** Le serveur utilise des tubes nommés pour recevoir les demandes des clients. Il attribue ensuite chaque demande à un thread disponible pour traitement.
5. **Utilisation de Mémoires Partagées :** Les threads de traitement utilisent des mémoires partagées pour communiquer avec le processus principal et pour stocker les résultats des traitements.
6. **Nettoyage et Fermeture :** Une fois que toutes les demandes ont été traitées ou que le serveur reçoit un signal SIGINT, il termine proprement en fermant tous les tubes nommés et en attendant que tous les threads se terminent.

Étapes dans le Programme d'un Client :

1. **Envoi de la Demande au Serveur :** Le client envoie une demande au serveur via un tube nommé pour obtenir un thread disponible pour le traitement.
2. **Réception de la Réponse du Serveur :** Le client crée un tube nommé individuel pour recevoir la réponse du serveur. Il attend ensuite la réponse du serveur via ce tube.
3. **Utilisation de la Mémoire Partagée :** Si un thread est disponible du côté du serveur, le client crée et utilise une mémoire partagée pour communiquer avec le thread attribué par le serveur. Il écrit une commande saisie par l'utilisateur dans la mémoire partagée et attend la réponse du thread.

En suivant ces étapes et en utilisant les solutions de communication décrites, nous assurons un fonctionnement fluide et fiable du lanceur et de ses clients, permettant ainsi une interaction efficace entre eux.

Dans les programmes fournis, les sémaphores, les mémoires partagées et les tubes sont nommés de la manière suivante :

Pour les Sémaphores :

Dans les deux programmes (serveur et client), les sémaphores sont nommés en fonction de l'identifiant du thread ou du processus client auquel ils sont associés. Ceci est réalisé en concaténant l'identifiant du thread ou du processus client avec des chaînes de caractères spécifiques pour former des noms uniques pour chaque sémaphore.

Exemple (pour un thread avec l'identifiant `1`) :

- Sémaphore de lecture : "/Read_1"
- Sémaphore d'écriture : "/Write_1"

Pour les Mémoires Partagées :

Dans le programme serveur, les mémoires partagées sont nommées en fonction de l'identifiant du thread associé. Ceci est réalisé en ajoutant l'identifiant du thread à une chaîne de caractères spécifique pour former un nom unique pour chaque mémoire partagée.

Exemple (pour un thread avec l'identifiant `1`) :

- Mémoire partagée : "/Memory_Thread_1"

Pour les Tubes :

Dans le programme client, les tubes sont nommés en fonction de l'identifiant du processus client associé. Ceci est réalisé en ajoutant l'identifiant du processus client à une chaîne de caractères spécifique pour former un nom unique pour chaque tube.

Exemple (pour un processus client avec l'identifiant 1234) :

- Tube de réponse : "/tmp/1234"

Dans le programme serveur, le tube pour recevoir les demandes des clients est nommé de manière fixe : "/tmp/tube_demande".

Cette approche garantit que chaque ressource (sémaphore, mémoire partagée, tube) est associée de manière unique à chaque thread ou processus client, ce qui évite les conflits et assure un fonctionnement correct du programme dans des environnements multitâches ou multi-utilisateurs.

3. Conclusion

En conclusion, les programmes serveur et client présentent une architecture robuste pour la communication entre un lanceur et ses clients. Voici quelques points clés à retenir :

1. **Communication Multi-Processus :** Les programmes utilisent des mécanismes de communication inter-processus tels que les tubes nommés et les mémoires partagées pour permettre une communication efficace entre le serveur et ses clients.
2. **Gestion des Threads :** Le serveur crée et gère plusieurs threads pour traiter les demandes des clients de manière concurrente, ce qui permet un traitement efficace des demandes entrantes.
3. **Synchronisation et Sécurité :** L'utilisation de sémaphores garantit une synchronisation appropriée entre les threads et les processus clients, évitant ainsi les problèmes de

concurrence et assurant un accès sûr aux ressources partagées.

4. **Flexibilité et Extensibilité :** Les programmes sont conçus de manière à être facilement configurables en fonction des besoins spécifiques. Les paramètres de configuration peuvent être ajustés dans le fichier de configuration, ce qui permet une adaptation facile à différentes situations.
5. **Fiabilité :** Les programmes sont conçus pour être robustes et résilients. Ils gèrent les signaux système tels que SIGINT de manière appropriée pour permettre une terminaison propre en cas d'interruption.

Ensemble, ces programmes offrent une solution complète pour la communication entre un lanceur et ses clients, permettant un échange efficace de données et une exécution fiable des commandes. Ils peuvent être utilisés dans une variété de contextes où une communication multi-processus est nécessaire, fournissant ainsi une base solide pour le développement d'applications distribuées.