2023-2024

TP 7 – un seul serveur mais plusieurs clients : gérer la concurrence

Exercice 1 : convertir l'application « comptage et tri de requêtes » pour TCP
Reprendre les codes de l'exercice 2 du TP5 et les convertir pour qu'ils fonctionnent avec TCP à la place d'UDP.

Exercice 2 : application « comptage et tri de requêtes » multi-threadée
On souhaite modifier le serveur de l'exercice 1 ci-dessus pour en faire une version multi-threadée capable de gérer plusieurs clients en même temps (ou successivement). On va procéder par étapes.
Note : le client n'a pas besoin d'être modifié.

E1 : conversion pour un seul client
Modifier le serveur pour qu'il gère tous les échanges avec un seul client dans un thread secondaire. Cela signifie que toutes les actions après acceptation du client (i.e. tous les échanges avec le client) sont exécutées dans un thread secondaire. Celui-ci fonctionne tant que le critère d'arrêt (i.e. le message 'fin') n'est pas reçu. Le thread principal s'arrête quand le thread secondaire prend fin.

E2 : extension pour plusieurs clients
Plusieurs difficultés sont à prendre en compte :

- ✔ la version E1 ne comporte pas de boucle dans le thread principal car il n'accepte qu'un seul client. Il faut donc créer une boucle dans le thread principal de manière à accepter plusieurs clients. La sortie de cette boucle doit mettre fin au serveur. Mais le serveur ne doit pas s'arrêter avec la réception d'un message de fin (sinon, des clients en cours d'exécution se retrouveraient avec une déconnexion sauvage du serveur). Il faut donc définir un critère de sortie de boucle indépendant des échanges avec les clients. Pour simplifier, on propose de limiter l'exécution du serveur au traitement de N clients (simultanément et / ou successivement). Le serveur doit s'arrêter après la fin des N threads secondaires
- ✔ il faut gérer une liste de sockets (1 socket supplémentaire par client)
- ✔ il faut gérer une liste de threads (1 thread secondaire par client)
- ✔ chaque client doit avoir un décompte de requêtes propre à lui

Exercice 3 : application « comptage et tri de requêtes » multiplexée

On souhaite modifier le serveur de l'exercice 1 ci-dessus pour en faire une version multiplexée capable de gérer plusieurs clients en même temps (ou successivement).

Note : le client n'a pas besoin d'être modifié.

Le serveur ne doit comporter qu'une seule boucle incluant à la fois l'acceptation d'un nouveau client et / ou un seul échange avec un client quelconque (déjà connecté). En effet, on ne peut monopoliser le serveur pour des échanges en boucle avec un seul client.

La sortie de boucle ne peut, une fois de plus, être liée aux échanges C / S. On va utiliser le même type de critère simpliste qu'à l'exercice 2 (donc différente de celle de l'exemple du cours). Mais la nouvelle architecture modifie la logique :

- il faut compter le nombre de clients qui ont fini les échanges ; le serveur s'arrête lorsqu'on a atteint le max
- il faut aussi compter le nombre de clients qui se sont connectés depuis le lancement du serveur (et pas à un instant T) ; une fois le max atteint, il faut rejeter les nouvelles demandes i.e. accepter et arrêter immédiatement les échanges (au niveau du client, on aura un arrêt « sauvage »)

Indications :

- ✔ comme dans l'exercice 2, il faut gérer une liste de sockets (1 socket par client)
- ✔ le décompte de requêtes est forcément global => il faut gérer une liste de compteur en parallèle à celle des sockets
- ✔ globalement, on peut créer une fonction reprenant l'essentiel de la méthode de thread de l'exercice 2, en y supprimant la boucle, mais avec un passage de paramètre lié au compteur