INFOF-201: Systèmes d'opération

Projet n°2: Programmation système

Introduction

Problématique

Le but de ce projet était de réaliser une application client-serveur mettant en place un salon de chat. Ceci dit certaines conditions étaient imposées: un nombre illimité de connexions, chat entre tous les utilisateurs connectés sur le serveur avec le pseudonyme et l'heure de l'envoie devant les messages et finalement une bonne gestion des arrêts inopportuns.

Approche différentes du problème

Avec les connaissances acquises en Tp, beaucoup de choix s'offraient à nous au niveau des implémentations pour gérer pour le client de recevoir et d'envoyer des messages à tout moment, et pour le serveur d'accepter des utilisateurs et envoyer à tous les utilisateurs les messages reçus.

Alors ma première approche a été dès le début d'utiliser 2 threads pour le client, 1 pour recevoir les messages et l'autre pour en envoyer.

Pour ce qui est du serveur ma première approche était la fonction select() mais j'ai malheureusement abandonné celle-ci car j'ai été bloque à cause d'un bug causé par problème n'étant pas lié au select mais ne trouvant pas la source j'ai décidé de complètement changer mon orientation (le code écrit pour le select est encore en commentaires dans le code afin de pouvoir vous poser des questions pour mieux comprendre son utilisation).

J'ai ensuite décidé d'aborder les processus mais j'avais oublié le processus père ne partage pas ses variables avec tous ses fils ce qui m'aurait obligé à mettre beaucoup de variables globales ce qui ne me semblait guère propre.

J'ai donc finalement décidé de me pencher vers ma dernière option qui était de créer un thread pour chaque nouveau client et s'occuper de la distribution des messages ds chaque thread.

Mise en place des stratégies choisies

Client

Donc comme expliqué ci_dessus, la problématique a été résolue pour le client avec 2 threads, mais avant de lancer ces dux threads, afin d'éviter des problèmes asynchrones avec le thread du serveur, j'ai deux appels systèmes synchrones avec appels systèmes du serveur qui permettent la demande du pseudonyme proprement. Ces appels sont recv() pour recevoir le premier message du serveur qui demander d'introduire le pseudonyme et le deuxième est un send() qui permet de lui renvoyer la réponse.

On crée/lance ensuite les deux threads:

pthread_create(sender/receiver&,NULL,sending/receiving,(void*) socket

Les deux fonctionnent avec le même principe, une boucle while(connected), que tant qu'ils sont connectés effectue soit recv() ou send().

Pour tous les send effectués dans le client, on vérifie que la chaine de caractères envoyée n'est pas vide en examinant la longueur de ladite chaine.

J'utilise également dans le thread sending, un fflush(stdout) pour vider le stream de sortie car dans le cas où l'on reste dans la boucle il est nécéssaire de vider les valeurs précédentes.

Serveur

Comme dit au début du rapport, la solution choisie pour le serveur a été de créer un thread pour chaque nouveau client.

Variables globales importantes:

- int list_sockets[100], qui va contenir tous les file desciptors des nouveaux clients
- int availabilities[100] = {1}, va permettre de savoir si une place dans la list_socket est déjà employée ou non pour exploiter au mieux celle-ci.
- int sockets_counter = 0, garde le nombre maximum d'emplacement remplis dans list_sockets et availabilities.

En premier lieu, pour chaque nouvelle connection au serveur on va tout d'avord vérifié avec la liste availabilities si une place s'est “libérée” (= un client s'est déconnecté) et si c'est le cas prendre sa place et dans le cas aucune n'a été “libérée” on lui donne alors une nouvelle place dans list_sockets, à la position du socket_counter.

Le fonctionnement du serveur est le suivant, on prépare un socket sur lequel on va pouvoir écouter et dès qu'on essaye de s'y connecter on crée un nouveau thread avec comme paramètre l'indice du file

decriptor du nouveau client dans la list_sockets[].

pthread_create(&thread,NULL,distribution,(void *) &indice_socket)

La fonction distribution va elle également bouclé sur le principe de toujours pouvoir recevoir un message mais avant cela va faire 2 appels systèmes symétriques à ceux fait chez le client afin de lui envoyer (send()) la demande du pseudonyme et de recevoir (recv()) sa réponse.

On rentre ensuite dans une boucle while(connected), qui va toujours lancer un appel bloquant recv() qui va attendre l'arrivée d'un message, et ensuite une deuxième boucle while qui va s'occuper de parcourir toute la list_socket jusqu'au sockets_counter, en vérifiant que l'élément est bien occupé grâce à availabilitieset lui renvoyer alors le message qu'il a reçu avec le pseudonyme de l'utilisateur et l'heure de la réception sur le serveur.

Conclusion

Déçu du résultat, je n'ai pas réussi à créer un input sans l'afficher sur le terminal, ce qui rend celui-ci peu agréable à manipuler mais correct malgré tout.

Le client et le serveur gèrent les fermetures inoppurtunes dans presque tous les cas, je viendrai vous poser des questions pour mieux comprendre la gestion de ce problème.

Globalement ce projet était très riche au niveau apprentissage et très intérssant afin d'apprendre à mélanger threads et réseau.