Un serveur HTTP minimaliste
Compte Rendu

Jérémie DUMAS

Table des matières

1 Un serveur multiclient

1.1 Gestion d'une pool de thread

Le programme crée au démarrage un certain nombre de thread pour gérer plusieurs clients en parallèle. Le paramètre est stocké dans une variable globale `max_thread_number`. La gestion des connections est alors faites ainsi :

- Le thread principal crée le socket serveur sur lequel il écoute
- Le thread principal s'occupe de faire un `accept` dès qu'il voit un nouveau client
- Il lève alors un sémaphore "un nouveau client est arrivé"
- L'un des threads en attente va baisser le sémaphore et récupérer le client
- Il lève alors un autre sémaphore "client récupéré", permettant au thread principal de se remettre en attente d'un autre client

On aurait pu envisager de créer en plus de cela d'autre thread au cours de l'exécution. La question est de savoir si cela vaut le coup d'un point de vue performance. En tout cas il n'y aurait pas grand chose à modifier pour ajouter cette fonctionnalité.

1.2 Utilisation de `pselect`

Un thread sera capable de s'occuper de plusieurs clients à la fois grâce à l'utilisation de `pselect`. On stocke les descripteurs de fichiers dans un tableau alloué satiquement (ça se passe dans `client.c`). L'idée générale est la suivante :

- Si un thread n'a pas de client, il va faire un `sem_wait` sur l'arrivée de nouveaux clients.
- Sinon, il fait juste un `sem_try_wait` mais ne se bloque pas s'il n'y a pas de nouveaux clients : on en a déjà d'autres à traiter.
- On fait alors un `pselect` avec timeout pour déterminer s'il s'est passé quelque chose parmi les clients que l'on surveille (événément en lecture ou en écriture).
- S'il ne s'est rien passé, on refait un tour de boucle. Sinon, on traite ce que l'on a à traiter.

A priori donc l'utilisation de `pselect` rendrait inutile l'utilisation de plusieurs thread, mais bon disons que c'est pour le sport.

A noter aussi que malgré l'utilisation de `pselect`, on attendra la réception de toute la requête Http avant de la parser. En revanche, pour l'envoi de données, on envoie d'abord les header, puis on utilise la fonction `fwrite` à chaque fois que `pselect` nous en donne l'autorisation, pour être sûr de ne pas bloquer.

Un petit inconvénient cependant : chaque client possède plusieurs buffer internes, qui servent au stockage d'une adresse, mais aussi des données à envoyer. Ainsi l'utilisation d'objets de type `FILE *` pour l'envoi de données double inutilement les buffer nécessaires. L'utilisation de `sendfile` a été envisagée, mais il a semblé que les performances n'étaient pas trop mauvaises tel quel, donc nous avons laissé ainsi.

2 Le serveur HTTP

2.1 Des requêtes simples

Le fichier qui s'occupe de lire les requêtes et d'y répondre correctement se nomme très justement `http.c`. Bon, il n'est pas extraordinaire en soit et ne reconnaît que les requêtes de type Get utilisant la version 1.1 de Http.

Mais bon c'est déjà sympa, et en plus il envoie une erreur 404 quand il ne trouve pas.

2.2 Journalisation

On remarquera qu'à l'exécution le serveur produit un joli petit fichier de journalisation, nommé par défaut `log`. Il ne raconte pas grand chose, mais ça montre au moins qu'il fonctionne. En tout cas l'écriture des données est protégée par un bon vieux mutex des familles, et ça c'est cool.

2.3 Ajout de types MIME

Le programme reconnaît certains types MIME les plus courants. On pourra l'essayer avec des images `png` ou `jpg` par exemple. Pour ajouter d'autres types MIME, on pourra modifier le fichier de configuration `mime.types` se trouvant dans le dossier `conf/`

2.4 Fichier de configuration

La cerise sur le gâteau, on peut configurer certains paramètres en modifiant le fichier de configuration `data.conf` du dossier `conf/`. Les paramètres sont assez explicites et ne seront donc pas détaillés outre mesure.

Notons tout de même qu'il y a d'autres paramètres importants qui ne sont pas accessibles via ce fichier de config' : par exemple la taille par défaut des buffers associés à chaque client, ou le timeout donné à la fonction `pselect`, donc on n'a pas encore pu mesurer l'impact véritable sur le serveur.

Conclusion

Le serveur que nous proposons, bien qu'étant assez minimaliste, a cependant divers avantages et inconvénients :

Il est assez léger, gère correctement les erreurs, et utilise de manière sûre les fonctions `pselect` avec une pool de threads.

D'un autre côté, il y a encore certaines vérifications nécessaires d'un point de vue sécurité qui ne sont pas effectuées : éviter de sortir du fichier racine dans l'url demandée, vérifier que le fichier à envoyer est bien lisible par `other`, etc.