Rapport du projet: Un serveur HTTP

Nguyen Phan Nguyet

4 Janvier 2017

Table des matières

1 Structure du serveur

1.1 Le serveur lancera une thread à chaque fois qu'un client se présente

D'abord, dans la fonction *main*, on crée une socket et écoute :

```
sc = socket (AF_INET, SOCK_STREAM, 0) ;
bind (sc,...) ;
listen (sc,...) ;
```

Pour chaque requete de connexion, *main* utilise la fonction *accept* et puis crée une thread qui sera chargée de cette connexion. La fonction *cn_func_thread* est donnée à *pthread_create* au moment de créer la thread. Le paramètre *exp* à passer à la thread est de la structure *th_sock* :

```
struct th_sock{
int sock;
struct sockaddr_in cli;
}
```

dont le champs *sock* pour contenir la descripteur de la socket associée à la thread (exp−>sock = accept (sc, &(exp−>cli))) et *cli* pour des infos du client qui a envoyé la requête de connextion.

1.2 Le serveur accepte un nombre de client de manière simultané

Après chaque *accept*, *main* décrémente le compteur *connec_count* (qui a été initialisé au nombre de client passé en ligne de commande) tant dis que chaque thread va l'incrémenter lors de la fermeture de la connexion. On utilise *mutex* pour synchroniser.

1.3 La réponse HTTP/1.1 nnn Message

La valeur *nnn* est récupérée à partir de la valeur de retour de la fonction *open* lors de la lecture du fichier.
fd = open (fichier, O_RDONLY)

- nnn = 200 si fd != -1 ou si le bit S_IXUSR est affecté dans le champs *st_mode* du *struct stat*
- nnn = 403 si errno == EACCES
- nnn = 404 si errno == ENOENT

2 Journalisation

2.1 Chaque requête provoque l'ajoute à la fin du fichier d'une ligne formée des informations

Cela est faite par l'ouverture du fichier par la fonction *fopen* avec l'option *a+* et l'écriture dans ce fichier par *fprintf*.
Les informations sont récupérées grâce à des fonctions : getpid(), stat(), inet_ntoa(), ctime().
Pour l'identifiant de la thread traitant la requête, il y a 2 cas :
-Si la requête est une exécution d'un exécutable, le serveur utilise le *pid* du fils.
-Sinon, il utilise le *tid* de la thread qui va traiter la requête.

2.2 Gérer les accès concurrents des Threads au fichier .log

On utilise *mutex*.

3 Fichier exécutable

S'il s'agit d'un exécutable, le serveur (P1) creé un processus fils (P2) qui va exécuter l'exécutable à travers *execl*. La communication entre P1 et P2 est faite par le tube.

3.1 Faut terminer au bout de 10 secondes, sinon être tué

Le P2 met *alarm(10)* avant de lancer l'exécutable à travers de *execv*. La fonction associée au traitement de SIGALRM est *kill (getpid())*. Cela assure que si l'exécution ne s'est pas terminée au bout de 10 secondes, P2 est tué.
Enfin, P2 exit par *exit (err)* où *err* est la valeur de retour de *execv* : err = execv(...). Si *exit* est exécuté, il y a donc une erreur dans l'exécution. Cette erreur peut être capturée par P1 via *waitpid*.

3.2 Communication entre P1 et le fils

P1 et P2 se communique à travers d'un tube. Dans le fils, la sortie standard est redirigée vers le tube. P1 appelle en boucle *waitpid* avec l'option *WNOHANG* pour lire sur ce tube la réponse de l'exécutable et puis l'écrit dans un fichier. Dans le but d'éviter des concurrences sur l'E/S sur ce fichier, le fichier est nommé par le nome du fichier demandé et le *pid* du fils et sera supprimé avant que le fils se termine. Ce fichier permettra aussi de récupérer la longeur de la réponse (par *stat()*) pour la journalisation ou pour la ligne *Content-Length* plutard .
Pour savoir s'il s'agit du cas où P2 n'a pas terminé l'exécutable au bout de 10 seconds ou du cas où la valeur de retour est différente de 0, il suffit d'appeler *WIFSIGNALED* ou de tester sur l'etat du fils.

4 Requêtes persistentes

4.1 Plusieurs requêtes dans une même connecxion. Chaque thread traite une requête.

Après avoir accepté une connexion d'un client, le serveur (*main*) crée une thread et lui passe en paramètre le descripteur du socket associé à cette connexion et des informations du client qui a demandé cette connexion (via *struct th_ sock*).
Ce que fait cette thread :
- Lire des requêtes du client.
- Envoyer la réponse "HTTP/1.1 nnn message".

- Faire journalisation.
- S'il s'agit d'un exécutable, elle crée un processus fils. Quand le fils se termine avec échec, elle fait à nouveau la journalisation.
- Sinon, elle crée une thread et lui passe en paramètre le nome du fichier, le descripteur du fichier et des informations du client (via *struct req_ arg*).

4.2 Transmettre les réponses dans l'ordre d'arrivée des requêtes

La thread qui est chargée de reçevoir des requêtes appelle *waitpis* ou *pthread_ join* avant de lire la prochaine requête.

4.3 Repérer la fin d'une réponse

- S'il s'agit d'un exécutable, la méthode est mentionnée dans la réponse 3.

Ce que on n'a pas fait dans cette question :
-Récupérer *Content-Type* dans l'exécutable.

5 Contrer le déni de services

On n'a pas fait cette question.