JANATI Siham
N° étudiant: 21914156

Résolution de Noms

Rapport de Projet

UE: Algorithmes des réseaux

I- Choix d'implémentation

Au départ, les serveurs sont lancés, une fonctions copient les données à partir des fichiers envoyé en argument dans des tableaux de structures "name_server", et retourne le nombre de serveurs.

```
struct name_server {
  int port;
  char *ip;
  char *nom;
}
```

Le client est lancé ensuite, une fonction stock les données à partir du fichier de serveurs racines envoyé en argument dans un tableau de structures "server", et retourne le nombre de serveurs racines.

```
struct server {
  char *ip;
  int port;
  int vitesse;
}
```

Les requêtes seront aussi copiées à l'aide d'une fonction (soit à partir d'un fichier, soit à partir de la ligne de commande) dans un tableau de structures "request", la fonction retournera le nombre de requêtes lues.

```
struct request {
  int id;
  int horodatage;
  char *racine;
  char *sous_domaine;
  char *domaine;
}
```

Toutes les adresses IPV4 seront mapées en IPV6 à l'aide d'une fonction de conversion avant d'être stockée dans la structure. Nous n'utiliserons que des sockets IPV6 pour simplifier le code. Le résultat final étant le même.

Tous les serveurs racines, et serveur de noms **pour les sous-domaines** executent le même code.
Ils reçoivent un nom de domaine de la part du client, et renvoient tous les serveurs qui traitent ce nom.
Le dernier serveur de nom appelé exécute un code différent, il n'envoit au client que le premier serveur trouvé.

Nous aurons donc 3 fichiers sources *pour un fonctionnement normal* : un pour le client, un pour les serveurs racines et les serveurs de sous-domaine, et un pour le dernier serveur de nom.

Pour les tests de pannes de serveurs et de temps d'attente, nous garderons les mêmes fichiers sources, en ajoutant quelques instructions côté serveur.

La primitive select nous permet de communiquer avec le serveur et detecter les temps d'attente important.
Avec plusieurs boucles imbriqués, le client traite chaque recvfrom avec un select, juste au cas où un serveur se présente comme performant au départ mais commence à avoir des délais de réponse important pour la suite des traitements. Dans ce cas, on passe au tour suivant de boucle pour communiquer avec le serveur suivant, et ainsi de suite jusqu'a ce qu'il n'y ait plus de serveurs disponible. Dans ce cas on quitte le programme vu que l'erreur vient de tous les serveurs, et que le client n'y peut rien.

Dans la structue *server*, nous avons une variable *vitesse* qui représente l'état du serveur, 1 si le serveur fonctionne normalement, 0 si il est trop lent.
Au départ, tous les serveurs ont leur variable *vitesse* à 1. Si un temps d'attente important pour un serveur est detecté, la variable *vitesse* est mise à 0.
Le client ne communique qu'avec les serveurs dont la variable *vitesse* est égale à 1.

Pour équilibrer la charge entre les serveurs racines, on utilise un tourniquet. A chaque tour de boucle, le client envoie la nouvelle requête au serveur suivant (avec la formule numéro-de-la-requête % nombre-de-serveurs-racines, qui nous donne le numéro du serveur avec lequel communiquer.)

II- Tests

a- Tourniquet

Fichier: tests/tourniquet.sh

Ce test lance une requête pour un nom de domaine traité par plusieurs serveurs de sous domaine.

Ce dernier utilise un tourniquet pour contacter **tous** les serveurs de sous domaine un par un, et stock leurs informations dans un tableau *server.* Ensuite, il contacte les serveurs du sous domaine suivant et s'arrête dès qu'il trouve une réponse.

Ce test donne l'exemple d'un fonctionnement normal du programme, sans délais d'attente de la part d'aucun des serveurs.

b- Cas où les serveurs racines sont tous très lent

Fichier: tests/elimination3.sh

Pour simuler des serveurs très lents, j'ai ajouté un sleep(2) dans le fichier racine_sleep.c

Le client passe par tous les serveurs et quitte le programme.

Par contre, si un seul serveur fonctionne correctement, le client communique avec ce serveur uniquement, comme est montré dans le test suivant.

c- Elimination d'un ou plusieurs serveurs pour délai d'attente important

-test1: tests/elimination1.sh certains serveurs ne répondent pas mais d'autre si
-test2: tests/elimination2.sh certains serveurs racines, et serveurs de noms ne répondent pas, sans autre option disponible pour résoudre le nom de domaine
-test3: tests/elimination2.sh aucun serveur racine ne répond

Une combinaison des tests a et b, où certains serveurs ne répondent pas tandis que d'autres répondent.
Le résultat ne changent pas si il y a d'autres serveurs qui peuvent les remplacer (test 1).
Mais dans le test 2, et le test 3, certaines requêtes n'auront pas de réponse.