Université de Strasbourg
UFR de Mathématiques et Informatique

Projet
Aide aux librairies

Etudiant : *Roussel Desmond Nzoyem*

UE : *Réseaux* – Enseignant : *Pr. Pierre David*
Date : *9 janvier 2021*

Résumé

Nous proposons une implemtation d'un système permettant de collecter les demandes des clients et les orienter vers des libraires ; chez qui les clients peuvent commander des livres et les chercher en mode "click and collect". Notre Implémentation repose sur le langage C et son API des sockets, suffisant pour implémenter les différentes options traitées dans ce rapport.

1. Description des protocoles

Les diffetents éléments du système sont : les **libraries**, le serveur **Nil**, et les **clients**. Ces trois eléments communiquent entre eux à travers des protocoles spécifiques.

1.1 Protocole Nil-Clients

Le client transmet une demande aux serveur Nil pour connaitre la disponibilité des certains livres. Le serveur répond avec une aggregation des différentes réponses des libraries qu'il a intérogé. On utilise le protocole TCP. Les datagrammes échangés suivent la convention indiquée à la figure 1.

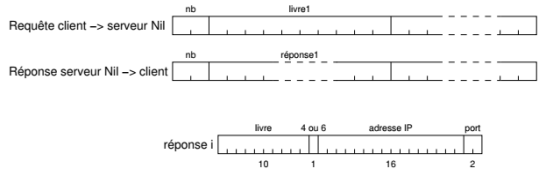

Figure 1 – *Protocole entre le serveur Nil et les clients.*

1.2 Protocole Nil-Librairies

Le serveur Nil retransmet la demande du client aux différentes libraries ; la librarie répond avec les livres qui sont disponibles. On utilise le protocole UDP ; ceci afin d'imposer un délai au serveur pour réponde aux requetes du client. Les datagrammes échangés suivent la convention indiquées à la figure 2.

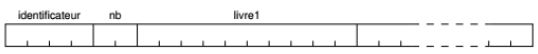

Figure 2 – *Protocole entre le serveur Nil et les librairies.*

1.3 Protocole Librairies-Clients

Le client commande un livre auprès de la librairie, la librarie confirme ou infirme la commande. On utilise le protocole TCP. Les datagrammes échangés suivent la convention indiquées à la figure 3.

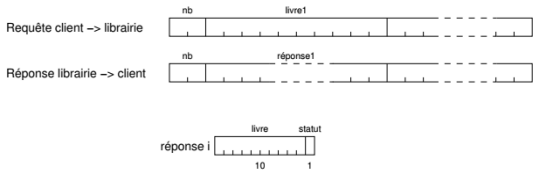

Figure 3 – *Protocole entre les librairies et les clients.*

2. Implémentation proposée

Nous avons suivi une approche **agile** pour construire le programme. Les fonctionnalités les plus faciles one été implémentées avant de passer aux plus compliqées. L'ordre suivi fut le suivant :

1. "envoire de requete" du client vers la librarie
2. "réception de requete" venant du client par la librarie

$\Longrightarrow$ On posside la version 1 du programme

3. "envoi de reponse" par la librarie au client
4. "reception de requete" venant du client par la librarie et affichage des resultats

$\Longrightarrow$ On posside la version 2

5. Communication Nil $\Longleftrightarrow$ librarie
6. Communication Nil $\Longleftrightarrow$ cleint

$\Longrightarrow$ On posside la version 3 finale

A la fin du codage, on obtient un programme effectuant les différentes tâches demandées. les fonctions de bibliiothèques `getaddrinfo`, `bind`, `connect`, et `select` sont regulièrement utilisées pour la construction de chacun des parties. En particulier, la fonction `getaddrinfo` sera implémentée pour traiter des addresses IPv6, mais pousrra aussi traiter des addresses IPv4.

2.1 Clients

Il s'agit d'un client TCP classique qui communique avec un serveur, le serveur pouvant être Nil ou une librarie ; La commande à executer est la suivante :

```
$ client serveur port livre1 livre2 ... livren
```

Les étapes principales de son implémentation, contenues dans le fichier `client.c`, sont les suivantes :

1. Récupération de l'addresse IP (IPv4 ou IPv6) du serveur à travers la fonction `getaddrinfo`
2. Création d'un socket pour se connecter au serveur (aucun `bind` n'est nécéssaire)
3. Ouverture active de connexion vers le serveur à travers la fonction `connect`
4. Envoi de la demande au serveur à travers la fonction `write_to_server`
5. Création d'une boucle infinie pour attendre la réponse du serveur a travers la fonction `select`
6. Lecture du contenu du message à travers la fonction `read_from_server`. C'est à ce niveau que la distinction entre un serveur de type Librarie ou Nil s'opère : le cleint lit l'octet numéro 13 de la réponse
   - Si ce byte contient la valeur 0 ou 1 : alors la réponse provient d'une librarie
   - Si ce byte contient la valeur 4 ou 6 : alors la réponse provient du Nil
   - Sinon, ce segment TCP n'est pas valide

2.2 Librairies

Il s'agit d'un serveur fonctionnant à la fois en TCP et en UDP. La commande à executer est la suivante :

```
$ librairie port livre1 livre2 ... livren
```

Remarque : *L'addresse IP de la librarie n'est pas fournie à la création. Cette adresse correspondra à addresse IP utilisée par l'interface (de la machine sur laquelle est lancée la librarie) pour communiquer avec le reste de l'internet*. Nous utiliserons généralement l'addresse de loopback `127.0.0.1` ou `::1` pour tester notre programme.

Le fichier correspondant se nomme `librarie.c`. Les étapes principales de l'Implémentation sont les suivantes :

1. Création du socket UDP pour la communication avec le serveur Nil
2. Bind du socket au port défini en argument
3. Création du socket TCP pour la communication avec le client
4. Bind du socket au meme port que précédemment
5. Se mettre en écoute du le port TCP pour aux plus `MAX_CLIENTS_NB` clients simultanément. Ceci à travers la fonction `listen`.
6. Création d'une boucle infinie pour attendre des connexions de Nil ou venant d'un client. Tout comme avec le client, nous devons utiliser la fonction `select`.
7. En cas d'un évènement de type "écriture" sur l'un des sockets :
   - S'il s'agit du socket UDP, alors traiter la demande du serveur Nil
   - S'il s'agit du socket TCP, alors faire une ouverture passive de connexion à travers la fonction `accept` ; ensuite traiter la commande du client (ne pas oublier de mettre à jour le stock de livres de la librairie)

2.3 Nil

la pièce centrale du système fonctionne à la fois en tant que client UDP et serveur TCP. La commande à executer est la suivante :

```
$ nil port délai librairie port librairie port...
```

Remarque : *Tout comme avec la librarie, l'addresse IP n'est pas fournie.*

Le fichier correspondant se nomme `nil.c`. Les étapes principales de son implémentation sont les suivantes :

1. Créer un unique socket TCP pour communiquer avec tous les clients
2. Bind du socket TCP au port défini en argument
3. Se mettre en écoute du le port TCP
4. Resoudre les addresses IP des différentes libraries
5. Creer autant de socket UDP d'il y a des libraries `nb_libs`. Le Bind du socket au port précisé pour le serveur Nil n'est pas nécessaire.
6. Creer une boucle infinie pour attendre des requetes clients, ou des réponses des libraries
7. En utilisant `select`, ecouter des évenment de type écriture sur les `1 + nb_libs` sockets surveillés, pendants au plus `délai` secondes :
   - Si le `select` s'achaève à cause de lárrivé d'un nouveau client, utiliser l'identifiant prévu pour le client, et retransmettre la requete du client vers toutes les libraries.
   - Si le `select` s'achaève à cause de la réponse librarie, identifier le client concerné par la réponse, et mettre a jour son buffer de réponse. Si on constate que ce client a été concerné déjà autant de fois qu'il y a de libraries, alors on peut renvoyer sa réponse ; mettre à jour le `next_id`, identifiant à utiliser pour le prochain client entrant [1]
   - Si le `select` s'achève par un timeout, alors tous les clients encore actifs ont attendu plus de `délai` secondes sans réponse. Renvoyer donc les réponses correspondantes à tous ces clients ; mettre à jour `next_id`.

3. Structures de donnnes

Les structures de données non trivialles mises en oeuvre sont les suivantes

- `uint8_t, uint16_t, uint32_t, et uint64_t` pour respecter les limitations dans chaque protocole

1. Le prochain identifiant à attribuer à un nouveau client par est les plus petit des identidiants devenus disponibles.

— pointeurs sur `uint8_t` pour les buffer qui von consituer les contenu des segments TCP, ou des datagrammes UDP échangés ;

— Pour le clien Nil

  — Des tableaus statiques pour sauvegarder les états des libraries (les addresses des libraires, sockets, etc.),dont le nombre est bien connu à l'avance

  — Des tablueax alloués dynamiquement pour sauvegarder les états des clients (actifs ou non, buffer de réponse, taille du buffer de réponse, etc.). Sachant qu'on peut avoir jusqu'a $2^32 - 1$ clients simulatanément, on décide d'allouer initialement de l'espace pour en traiter juste 32 ; en cas de nécessité, on etend cet espace.

4. Test visuel

NOus avons effectué des simulations sur un réseau local `192.168.188.0/24` specialement crée a cet effet [2]. La machine principale sur laquelle nous lancons le serveur Nil et le client a pour addresse IP `192.168.188.152/24`. On possede 4 librairies :

— La librairie n^o1 : `librairie 9001 B C A Zoro` est créée sur la machine principale, et contenant les livres B, C, A et Zoro

— La librairie n^o2 : `librairie 9002 Azimov Dune` est créée sur la machine principale, et contient les livres Azimov, et Dune

— Le librarie n^o3 : `librairie 9003 Zoro A` est créée sur une machine du réseau local `192.168.188.141/24`, et contient les livres Zoro, et A.

— Le librarie n^o4 : `librairie 9004` est créée sur la machine Turing `turing.u-strasbg.fr` et ne contient aucun livre

— Le serveur Nil :

```
\userinput{nil} \arguments{9000 5 127.0.0.1
9001 ::1
9002 192.168.188.141 9003 turing.u-strasbg.
fr 9004}
```

— Le librarie n^o4 : `librairie 9004` est créée sur la machine Turing `turing.u-strasbg.fr` et ne contient aucun livre

Les images de la simulation de la commande aupres d'un client sont aussi présentées

les fichier ayant permis de faire cette simulation sont :

— librarie.c

— ...

AFFICHER LES IMAGES

5. Limitations

Plusieurs sockets UDP sont utilisés (un socket pour chaque librarie), ce qui peut étre couteux si l'on a beacoup de libraries vers lesquelles on peut chercher. [3]. Avec suffisament de temps, on pourrait faire une boucle sur les sockets, et ne garder que deux sokets pour toutes les libraries.

2. La nécessité d'un réseau local vient du besoin d'éviter les parefeu, et les mots de passe necessaires pour se connecter en SSH par exemple.

3. Cette difficulté a été introduite par le fait qu'une addrsse IP utilisées pour se connecter a une librarie peuvent varier de l'IPV4 a l'IPV6