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*

Introduction

Nous proposons une implémentation d'un système permettant de collecter les demandes des clients et de les orienter vers des librairies, chez qui les clients pourront commander et chercher des livres en mode "click and collect". Notre implémentation repose sur le langage C et son API des sockets, suffisants pour faire fonctionner les différentes options traitées dans ce rapport.

1. Description des protocoles

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

1.1 Protocole Nil - Clients

Le client transmet une demande au serveur Nil pour connaitre la disponibilité des certains livres. Nil répond avec une agrégation des différentes réponses des librairies qu'il a interrogé. Ces deux parties communiquent à travers le protocole TCP. Les segments échangés suivent la convention indiquée à la figure 1.

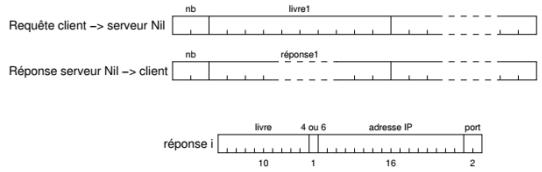


Figure 1 – *Protocole entre le serveur Nil et les clients.*

1.2 Protocole Nil - Librairies

Le serveur Nil retransmet la demande du client aux différentes librairies ; chaque librairie répond avec les livres qui sont disponibles. Ici, on utilise le protocole UDP afin d'imposer un délai au serveur pour répondre aux requêtes des clients. Les datagrammes échangés suivent la convention indiquées à la figure 2.

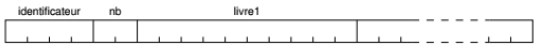


Figure 2 – *Protocole entre le serveur Nil et les librairies.*

1.3 Protocole Librairie - Client

Le client commande un livre auprès de la librairie ; la librairie confirme ou infirme la commande. Ici, on utilise le protocole TCP. Les segments échangés suivent la convention indiquée à la figure 3.

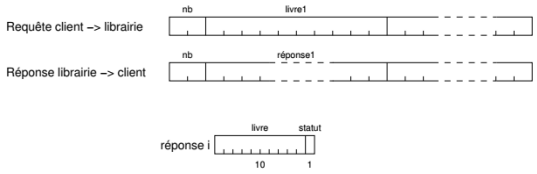


Figure 3 – *Protocole entre les librairies et les clients.*

2. Implémentation

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

1. "envoi de requête" du client vers la librairie
2. "réception de requête" venant du client par la librairie

$\Longrightarrow$ On possède la **version 1** du programme

3. "envoi de réponse" par la librairie au client
4. "réception de réponse" venant de la librairie par le client ; et affichage des résultats

$\Longrightarrow$ On possède la **version 2**

5. Communication Nil $\longleftrightarrow$ librairies
6. Communication Nil $\longleftrightarrow$ clients

$\Longrightarrow$ On possède la **version 3** finale

À la fin du codage, on obtient un programme effectuant les différentes tâches demandées. Les fonctions de bibliothèques `getaddrinfo`, `bind`, `listen`, `connect`, et `select` ont régulièrement été utilisées dans la construction de chacun des composants. En particulier, la fonction `getaddrinfo` a été utilisée pour "résoudre" les adresses IP, qu'elles soient en IPv6 ou en IPv4.

2.1 Clients

Il s'agit d'un client TCP classique qui communique avec un serveur, le serveur pouvant être Nil ou une librairie ; la commande à exécuter 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'adresse IP du serveur à travers la fonction `getaddrinfo` (protocole IPv6 pouvant supporter IPv4)
2. Création d'un socket pour se connecter au serveur (aucun `bind` n'est nécessaire)
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 à 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 "Librairie" ou un serveur de type "Nil" s'opère ; le client lit l'octet numéro 13 de la réponse :
   - Si cet octet contient la valeur 0 ou 1 : alors la réponse provient d'une librairie
   - Si cet octet contient la valeur 4 ou 6 : alors la réponse provient de 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 exécuter est la suivante :

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

Remarque : *L'adresse IP de la librairie n'est pas fournie à la création. Cette adresse correspond à l'adresse IP utilisée par l'interface (de la machine sur laquelle est lancée la librairie) pour communiquer avec le reste de l'internet.*

Le fichier correspondant se nomme `librairie.c`, et 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 pour cette librairie (donné en argument)
3. Création du socket TCP pour la communication avec le client
4. Bind du socket au même port que précédemment
5. Mise en écoute sur le port TCP pour au plus `MAX_CLIENTS_NB`[1] clients ; ceci à travers la fonction `listen`.
6. Création d'une boucle infinie pour attendre des connexions entrantes de Nil ou d'un client. Tout comme lors de l'implémentation du client, nous devons utiliser la fonction `select` à ce niveau.
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)

1. Attention, cette directive de préprocesseur est définie différemment pour la librairie et pour le serveur Nil.

2.3 Nil

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

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

Remarque : *Tout comme avec la librairie, l'adresse IP n'est pas fournie. Elle est déduite de la machine qui crée le serveur.*

Le fichier correspondant se nomme `nil.c`, et 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 pour le serveur
3. Se mettre en écoute sur le port TCP
4. Résoudre les adresses IP des différentes librairies
5. Créer autant de sockets UDP qu'il y a de librairies `nb_libs`. Un `bind` des sockets au port défini pour le serveur Nil n'est pas nécessaire.
6. Créer une boucle infinie pour attendre des requêtes de clients, ou des réponses de librairies
7. En utilisant `select`, écouter des événements de type "écriture" sur les `nb_libs+1` sockets surveillés, pendants au plus `délai` secondes :
   - Si le `select` s'arrête à cause de l'arrivée d'un nouveau client : utiliser l'identifiant prévu pour le client, et retransmettre la requête du client vers toutes les librairies. Sachant qu'un meilleur identifiant sera probablement libéré, prendre le `next_id`[2] comme étant le plus grand identifiant utilisé jusqu'à présent plus 1.
   - Si le `select` s'arrête à cause de la réponse d'une librairie : identifier le client concerné par la réponse, et mettre à jour son buffer de réponse. Si on constate que ce client a été concerné déjà autant de fois qu'il y a de librairies, alors on peut lui envoyer sa réponse ; ensuite il faut mettre à jour le `next_id` ; on prendra cet identifiant (devenu disponible) s'il est plus petit que le `next_id` actuel.
   - Si le `select` s'achève par un "timeout" : alors tous les clients encore actifs ont attendu plus de `délai` secondes sans réponse. Il faut donc renvoyer les réponses correspondantes à chacun de ces clients ; ensuite mettre à jour `next_id` ; on prendra le plus petit des identifiants devenus disponibles.

2. Il s'agit de l'identifiant à utiliser pour le prochain client entrant

3. Structures de données

Les structures de données non triviales mises en œuvre sont les suivantes :

- Les types `uint8_t, uint16_t, uint32_t, et uint64_t` pour respecter les limitations de chaque protocole, et avoir un code portable.
- Les pointeurs sur des `uint8_t` pour les buffers, qui vont constituer les segments TCP ou les datagrammes UDP échangés.
- Pour le serveur Nil :
  - Des tableaux statiques pour sauvegarder les états des librairies (adresses IP, descripteurs de sockets, etc.), dont le nombre est bien connu à l'avance.
  - Des tableaux alloués dynamiquement pour sauvegarder les états des clients (activité, buffer de réponse, taille du buffer de réponse, etc.). Sachant qu'on peut avoir jusqu'à $2^{32} - 1$ clients simultanément, on décide d'allouer initialement de l'espace pour n'en traiter que 32. En cas de nécessité, on étendra cet espace.

4. Test visuel

Nous avons effectué une simulation sur un réseau local `192.168.188.0/24` spécialement créé à cet effet [3]. La machine principale (celle sur laquelle nous lançons le serveur Nil et le client) a pour adresse IP `192.168.188.152/24`. Les différents éléments de la simulation sont les suivants :

- La librairie #1 est créée sur la machine principale, et contient les livres B, C, A, et Zoro :

  ```
  $ librairie 9001 B C A Zoro
  ```

- La librairie #2 est créée sur la machine principale, et contient les livres Azimov et Dune :

  ```
  $ librairie 9002 Azimov Dune
  ```

- La librairie #3 est créée sur une machine du réseau local `192.168.188.141/24`, et contient les livres Zoro et A :

  ```
  $ librairie 9003 Zoro A
  ```

- La librairie #4 est créé sur la machine `turing.u-strasbg.fr` et ne contient aucun livre :

  ```
  $ librairie 9004
  ```

- Le serveur Nil est créé sur la machine principale et interroge les 4 librairies :

  ```
  $ nil 9000 5 127.0.0.1 9001 ::1 9002
  ↪ 192.168.188.141 9003 turing.u-strasbg.fr
  ↪ 9004
  ```

- Le client est créé sur la machine principale et interroge le serveur Nil :

  ```
  $ client ::1 9000 Dune Zoro Perl Taron A
  ↪ Azimov
  ```

Les résultats obtenus sont présentés à la figure figure 4. On y observe plusieurs points qui font la force de ce programme :

- Les système fonctionne correctement en IPv6 comme en IPv4. En réalité, il fonctionne en IPv6 et les adresses IPv4 sont convenablement converties en IPv6 (cf. figures 4c et 4e).
- La librairie #4 située sur `Turing` (d'adresse IP `130.79.255.76`) n'est pas accessible à partir du réseau local créé (cf. figure 4b), a moins d'établir une connexion SSH. Cela n'empêche pas le programme de fonctionner ; et une réponse est retournée au client après que le délai de 5 secondes s'est écoulé.

Les atouts de notre implémentation du serveur `nil`, qui ne sont pas directement observables depuis la figure 4 sont :

- Le serveur ne nécessite pas de `fork`, tout se passe dans le processus créé initialement ; ce qui facilite le débogage en cas de problème.
- Le serveur supporte jusqu'à `MAX_CLIENTS_NB = 4294967295` clients simultanément. Pour ce faire de manière efficace, on a recours à une attribution des identifiants de manière astucieuse (décrite à la section 2.3), qui permet d'économiser la mémoire RAM utilisée. Cela est d'autant plus important que le serveur Nil (tout comme les différentes librairies) peut tourner indéfiniment sans jamais s'arrêter.

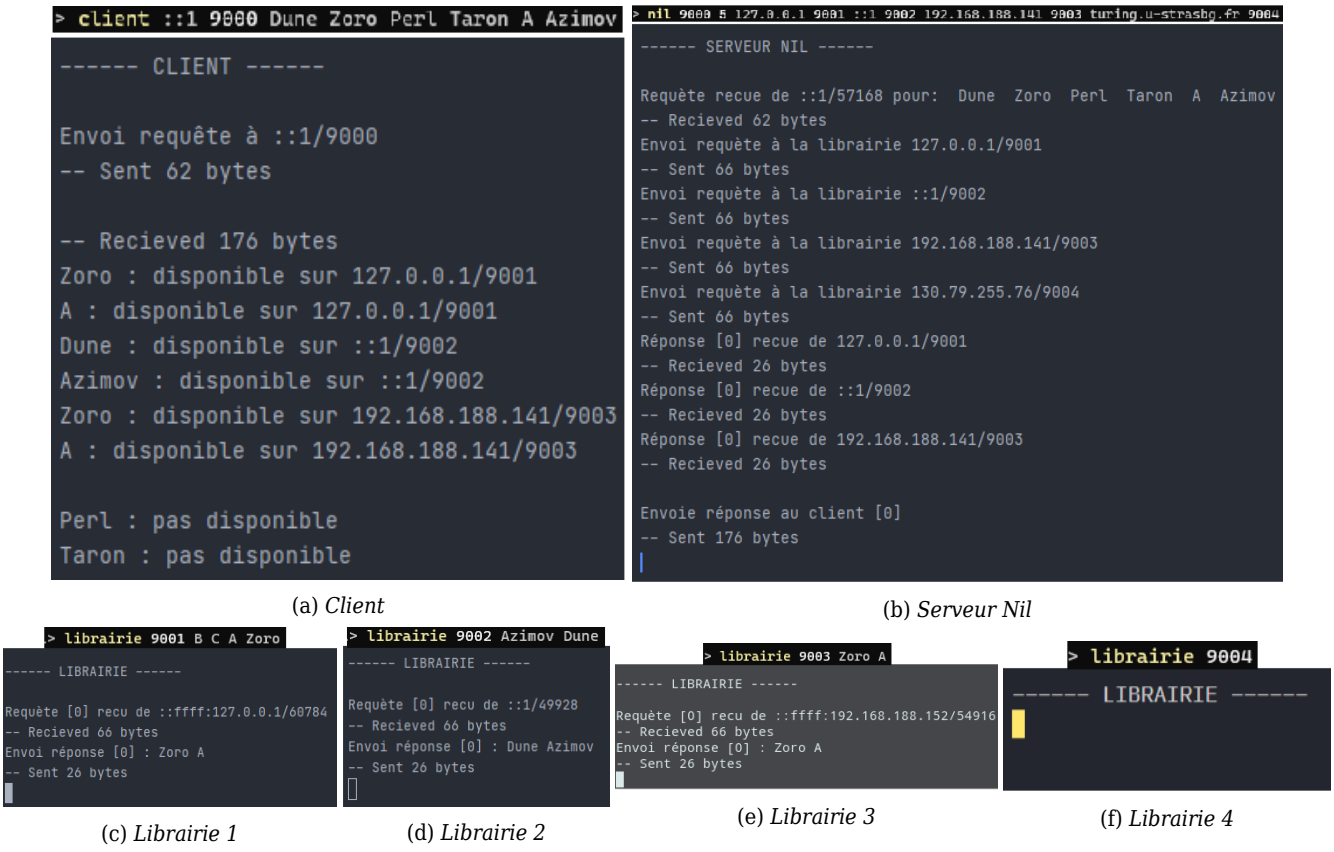
5. Limitations

Les principales limitations du programme sont les suivantes :

- **Plusieurs ports UDP** sont utilisés par le serveur pour communiquer avec les librairies (cf. figures 4c à 4e), ce qui peut être prohibitif si l'on a un grand nombre de librairies à interroger. Cette difficulté a été introduite par le fait que les adresses IP utilisées pour se connecter aux librairies peuvent varier (IPv4, IPv6, adresse IP secondaire) ; en plus, une adresse valide lors d'une première connexion peut ne plus être valide lors de la suivante. La résolution de ce problème nécessite un `bind` au port dédié au serveur Nil [4].
- Le **nombre de clients** qu'une librairie peut mettre dans sa file d'attente est raisonnablement limité à `MAX_CLIENTS_NB=65335`. Autrement dit, si plus de `65335` clients essaient simultanément de commander des ouvrages auprès de la même librairie, certaines commandes seront simplement ignorées.
- La **tolérance aux pannes** (qui consiste à émettre une nouvelle requête à destination d'une librairie si Nil n'a pas reçu de réponse après le délai imposé) n'a pas été implémentée. Le processus a cependant été initié. Pour le mener à terme, il faudrait utiliser le tableau `active_clients` qui indique l'activité de tous les clients :
  - Si `active_clients[j] = 0`, alors, de l'espace a été alloué pour recevoir le client `j`, mais celui-ci n'existe pas encore ; ou alors ce client à déjà entièrement été traité.
  - Si `active_clients[j] = 1`, alors le client `j` a attendu pendant `délai/2` secondes, mais n'a toujours pas reçu la totalité des `nb_libs` réponses qu'il a besoin. Il faut donc renvoyer une requête aux librairies concernées par le retard.

3. La nécessité d'utiliser un réseau local vient du besoin d'éviter les pare-feux et les mots de passe nécessaires pour se connecter en SSH par exemple.

4. En IPv6, une liaison (ou "bind") au port dédié a pu être établie ; cependant en IPv4, des difficultés ont été rencontrées.



(a) *Client* (b) *Serveur Nil*

(c) *Librairie 1* (d) *Librairie 2* (e) *Librairie 3* (f) *Librairie 4*

Figure 4 – *Affichage de chacune des parties lors de la simulation du fonctionnement du système. Les commandes exécutées par chacune des parties, reprises ici, sont détaillées à la section 4. Ces résulats indiquent que le client peut décider de faire, avec succès, une commande auprès des librairies 1, 2, ou 3. C'est cela qu'on observe à la figure 5.*

(a) *Client* (b) *Librairie 1*

Figure 5 – *Simulation d'une commande de deux livres par un client auprès d'une librairie. Cette simulation complète celle observée à la figure 4, et permet d'observer le protocole Librairie - Client à l'œuvre. Notons que cette simulation n'a pas été faite immédiatement après la fin de la requête auprès de Nil. Sinon, on aurait observé un affichage continu du côté de la librairie, et le port attribué au client par l'OS aurait sans doute été le même.*

— Si `active_clients[j] = 2`, alors le client `j` a attendu pendant `délai` secondes, mais n'a toujours pas reçu ses `nb_libs` réponses. Dans ce cas, il faut immédiatement renvoyer la réponse au client.

On constate que pour résoudre ce dernier problème, il faut garder en mémoire non seulement la requête que chaque client a effectué auprès du serveur Nil, mais aussi quelles librairies n'ont pas répondu à la demande de Nil.