TP Client/Serveur – GL2 : Utilisationde l'interface *sockets* pour des communications TCP/UDP de type client/serveur

Objectifs

- utilisation de l'interface sockets
- développement d'un client HTTP
- développement et études d'une application Client/Serveur en mode connecté
- développement et études d'application Client/Serveur en mode non connecté
- développement d'un serveur traitant des requêtes de plusieurs clients (mode concurrent)

Pré-requis

Programmation en langage C, notions sur TCP/UDP

NB

Ce TP nécessite un travail personnel conséquent (20 à 30h).

1. Introduction (pour vous aider !)

- Outils de capture de paquets :

Vous aurez à utiliser au cours de ce TP un outil de capture de paquets (analyseur réseau). Pour communiquer, les machines échangent des informations sous forme de paquets qui sont l'unité de données échangées sur le réseau. Il est possible « d'écouter » le câble Ethernet et de regarder ce qui se passe quand vous lancez des commandes comme `ping`, `rlogin`, …

Un premier outil permettant d'observer le réseau s'appelle `tcpdump`. Pour écouter le trafic sur l'interface `eth0`, il faut taper la commande `tcpdump -i eth0`.

Par défaut, `tcpdump` écoute en « *promiscuous mode* », c'est à dire qu'il capture et analyse toutes les trames circulant sur le réseau même celles qui ne concernent pas la machine sur laquelle il tourne.

Un deuxième outil, un peu plus convivial que `tcpdump` car utilisable en mode graphique avec un affichage plus lisible, est l'utilitaire `wireshark`. N'oubliez pas de cocher « `Update list of packets in real time` ».

Ces deux outils permettent d'établir des filtres de capture de paquets de manière assez fine. La syntaxe des filtres utilisés est la même pour `wireshark` que pour `tcpdump`. Par exemple, la commande `tcpdump -i eth0 tcp dst port 53` permet de ne capturer sur l'interface `eth0` que les segments TCP dont le port de destination est 53. Pour plus d'informations, consultez la page manuelle de `tcpdump`.

2. Développement d'un client HTTP

2.1. Vérification de la présence du serveur HTTP

Le format d'une requête HTTP est le suivant :
- la première ligne est composée d'une commande HTTP (GET/POST/HEAD/…), d'une URL qui identifie la ressource demandée puis de la version du protocole HTTP utilisée
- les lignes suivantes constituent l'en-tête de la requête HTTP
- la fin de l'entête est signalée par une ligne vide
- les lignes suivantes constituent le contenu de la requête qui bien souvent est vide (sauf dans le cas d'une requête POST)

Voici un exemple de requête qui demande des informations sur la page d'accueil du serveur HTTP à l'aide de la commande HEAD. La réponse du serveur se trouve à la suite de la requête.

```
ogluck@lima:~$ telnet 10.250.101.1 80
Trying 10.250.101.1...
Connected to localhost.
Escape character is '^]'.
HEAD / HTTP/1.1
Host: localhost
Accept: */*

HTTP/1.1 200 OK
Date: Mon, 13 Nov 2017 12:31:23 GMT
Server: Apache/2.4.10 (Debian)
Content-Type: text/html; charset=iso-8859-1

Connection closed by foreign host.
ogluck@lima:~$
```

Manipulation

Contactez le serveur HTTP sur le port 80 (port standard HTTP) de la machine serveur (10.250.101.1) et effectuez une requête analogue pour vérifier que le serveur est en bon état de marche. Quelle commande utilisez-vous ? Remplacez le HEAD par un GET. Que constatez-vous ?

2.2. Client HTTP en mode connecté

Manipulation

Ecrire un programme client en mode connecté qui lit au clavier une requête HTTP et affiche la réponse du serveur.

Manipulation et questions

Une fois que votre programme fonctionne correctement, regardez avec un outil de capture de paquets les messages HTTP, les segments TCP et les datagrammes IP échangés au cours d'un échange Client/Serveur. Comment s'établit la connexion au niveau HTTP et au niveau TCP ? Qui décide de fermer la connexion ? Quand ? Quels sont les ports utilisés par le client et le serveur ? Dessinez un chronogramme des échanges observés.
Refaites la même capture en remplaçant votre client HTTP par celui de la question 2.1. Que constatez-vous ? Expliquez !

3. Transfert de messages en mode connecté

Manipulation

Ecrire un programme Client/Serveur qui transfère des messages en mode connecté (TCP) entre deux stations selon le schéma ci-dessous. La station 2 doit envoyer 60 fois l'heure courante à la station 1.

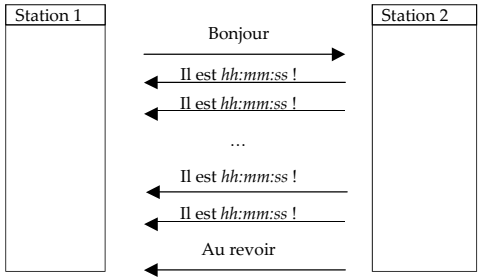
Manipulation et questions

- Comptez sur le client et sur le serveur le nombre de messages échangés. Refaites le même comptage en ajoutant sur le serveur un délai d'environ une seconde entre chaque message (`sleep`). Que constatez-vous ? Y a t-il des pertes de messages ?
- Regardez avec `wireshark` les messages échangés. Les segments TCP envoyés correspondent t-ils exactement aux écritures que votre programme fait sur la socket ?
- Que se passe t-il si vous débranchez le câble réseau reliant le client et le serveur ? Même question si vous débranchez puis rebranchez rapidement le câble. ?
- Pouvez-vous avec votre implémentation actuelle servir plusieurs clients ? Si oui, peuvent-ils être servis simultanément ? Lancez simultanément un nombre suffisant de clients pour remplir la file des connexions pendantes du serveur. Que se passe t-il si vous lancez alors un client supplémentaire alors que la file est pleine ?

4. Transfert de messages en mode non connecté

Manipulation

Ecrivez une nouvelle version du programme précédant en mode non connecté (UDP). Refaites les manipulations demandées et répondez de nouveau aux questions posées. Quelles sont vos conclusions ? Comment faire en sorte que votre serveur puisse répondre à plusieurs clients ?



5. Serveur en mode concurrent

Manipulation

Transformez le serveur en mode connecté écrit précédemment (avec le `sleep`) pour qu'il puisse répondre à plusieurs clients simultanément (serveur mono protocole, mono service en mode concurrent). Vous pourrez vous aider des exercices vus en cours. Mettez en place et décrivez un test permettant de constater que les multiples requêtes clientes sont bien traitées en parallèle. Que constatez-vous par rapport à l'écoulement du temps si le nombre de clients est important ?

Manipulation

Transformez maintenant le serveur afin qu'il puisse répondre à ***trois*** *services simultanément (serveur mono protocole, multi services en mode concurrent).*
Le serveur pourra conserver le service initial et proposer de nouveaux services de votre choix (exécution d'une commande distante par exemple pour connaître le nombre de processus qui tournent sur le serveur, transfert de fichiers, …). Mettez en place une procédure de test permettant de vérifier que deux requêtes clientes pour un service distinct sont bien traitées en parallèle.