TP n° 3 - 2ème année - RSA Réseaux

2016-2017

TP Programmation Avancée

L'ensemble des sources dont vous avez besoin pour votre TP sont à votre disposition à l'URL suivante : `https://members.loria.fr/IChrisment/files/Templates.tar.gz`. Le répertoire `Templates` est composé de 4 sous-répertoires :

- Select
- Raw
- Urgent
- IPv4

Le TP se déroulera en deux séances (la deuxième séance est facultative). A la fin de chaque séance, vous enverrez à votre encadrant les exercices terminés et vous supprimerez localement les fichiers que vous avez écrits.

Exercice 1

Dans le répertoire **Raw** se trouvent les sources d'un client UDP qui envoie l'heure à un serveur, attend un message du serveur et quitte l'application. Développer le serveur qui affiche l'heure reçue, renvoie la donnée reçue et se met ensuite en attente sur la requête d'un autre client.

Exercice 2

L'objectif de cet exercice est de vous faire manipuler le concept d'entrées/sorties multiplexées.

- Tester tout d'abord la version du serveur echo multi-client : **Select/servmulti_tcp.c**. Pour la partie client, vous pouvez utiliser **telnet**.
- Modifier cette version pour en faire un serveur multi-clients utilisant la primitive **select()**. Il s'agit d'implanter les notions vues en TD.

Exercice 3

L'objectif de cet exercice est de vous faire manipuler le concept de socket raw.

- Tout d'abord copier le fichier **Raw/icmpd-incomplet.c** dans **icmpd.c** et complétez-le
- Tester les différents scénarios permettant l'arrivée de messages ICMP de type ICMP_ECHOREPLY, ICMP_UNREACH et ICMP_TIMXCEED. Vous pourrez également vérifier la réception des messages ICMP via **wireshark**.

Exercice 4

L'objectif de cet exercice est de vous faire manipuler le concept de donnée urgent appelée aussi message OOB (Out Of Band).

- Tout d'abord copier le fichier **Urgent/serverUrgentTCP-incomplet.c** dans **serverUrgentTCP.c** et complétez-le.
- Exécuter le serveur en connectant le client dont le fichier source est **Urgent/clientUrgentTCP.c**. Analyser et expliquer le comportement du serveur.
- Observer avec **wireshark** l'en-tête TCP et notamment les informations relatives aux données urgentes.

Exercice 5

L'objectif de cet exercice est de vous faire manipuler le concept de multicast/communication de groupe.

- Ecrire le code de l'émetteur `sender.c` qui enverra la date courante à une adresse multicast. Il suffit pour cela d'adapter le code source **Raw/clientUDP.c**.
- Ecrire le code du récepteur qui affiche la date courante reçue sur cette adresse multicast.
- Tester votre application en lançant plusieurs récepteurs sur une même machine (voire sur d'autres machines).

Exercice 6
L'objectif de cet exercice est de vous faire manipuler les adresses IPv6.

— Porter en IPv6 le code relatif aux sockets TCP/IPv4 qui se trouve dans le sous-répertoire IPv4. Pour vous aider, des informations sont également disponibles le site de `http://livre.g6.asso.fr/`, dans la section Programmation d'applications. Pour le portage, deux versions sont demandées :

1. une première version qui prend en compte la modification des structures de données des adresses.
2. une deuxième version avec utilisation de l'appel getaddrinfo. Un exemple de client-serveur est donné sur le site. N'hésitez pas non plus à faire un **man de getaddrinfo**.

— Tester votre code sur le réseau local. Vérifier que la communication se fait bien en IPv6.

PROJET : MyAdBlock

Un serveur proxy HTTP est un serveur HTTP particulier qui relaie les différentes requêtes entre le client HTTP (i.e. firefox,...) et le véritable serveur web. Il est appelé aussi mandataire et permet entre autres de faire du filtrage au niveau applicatif (par exemple l'accès à certaines URL) et également de gérer un cache pour accélérer les temps de réponse.
La Figure 1 illustre le fonctionnement d'un proxy transparent c'est-à-dire, chargé uniquement de faire le relais entre le client et le serveur.

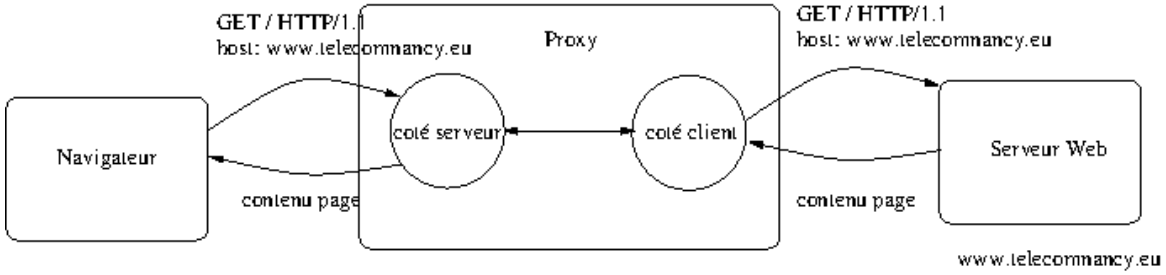

FIGURE 1 – Proxy web transparent

Dans le cadre de ce projet, il vous est demandé d'aller un peu plus loin en réalisant un proxy capable d'agir comme un bloqueur de publicité. Vous ne relayerez la requête que si le nom de l'hôte ne correspond pas à un serveur tiers hébergeur de publicités. Vous trouverez une liste de ces serveurs sur le site `https://easylist.to/`. Attention à la réponse à retourner au client si vous bloquez la requête.
Le travail devra être organisé en plusieurs étapes. Un dossier devra présenter et analyser ces différentes étapes.

1. Analyser les échanges TCP et HTTP entre un client et un serveur web (`www.telecomnancy.eu` par exemple) en vous aidant d'un analyseur de traces comme `wireshark`.
2. Configurer votre navigateur web pour définir un proxy HTTP.
3. Observer à nouveau, avec `wireshark`, le contenu des messages envoyés par le navigateur quand vous essayez de joindre votre serveur web.
4. Spécifier l'algorithme général de MyAdBlock permettant de gérer un client à la fois. Vous préciserez notamment comment le bloqueur détermine l'adresse du serveur à contacter. Vous vous intéresserez principalement à la commande GET.
5. Implanter MyAdBlock. Le test sera réalisé avec le serveur `http://www.01net.com/`. Pour faciliter le passage entre IPv4 et IPv6, vous utiliserez la primitive `getaddrinfo` pour obtenir l'adresse correspondant à une URL.
6. Étendre le proxy pour gérer plusieurs clients "simultanément".
7. Pour ceux ou celles qui veulent aller plus loin, étendre le proxy pour gérer des requêtes HTTPS (voir la méthode CONNECT permettant d'établir un tunnel sans déchifrrement) et tester avec `https://www.leboncoin.fr/annonces/`.

Le PROJET sera réalisé en binôme. Il est à rendre pour le **mardi 25 avril 2017**. Le rapport (format pdf) ainsi que les sources devront être déposés sous arche (format tar.gz). Aux sources devront être ajoutés un README et un makefile. Les soutenances seront organisées entre le jeudi 27 avril et le vendredi 28 avril.