PR6 – Programmation réseaux
TP n° 11 : Un client HTTPS

Exercice 1 : un client https

Le protocole HTTPS permet, entre autre, à un client de recevoir d'un serveur, de façon sécurisée, des données correspondant à certaines requêtes. Ce protocole est en particulier utilisé pour transmettre aux navigateurs (Firefox, Chrome, etc...) les documents HTML qui décrivent les pages web des sites Internet. Les échanges se font en TCP avec le protocole TLS pour les sécuriser. Le protocole HTTP permet de faire plusieurs types de requête suivant l'action que l'on souhaite effectuer : `GET`, `HEAD`, `POST`, `PUT`, etc...

Dans ce TP, on va écrire un client simple HTTPS qui ne traitera que des requêtes `GET`.

Requête `GET` Les requêtes `GET` servent notamment à récupérer les pages des sites web. Une requête débute par le message :

```
GET [page] HTTP/[version]\r\n
```

où `[page]` est la page demandée par la requête et `[version]` la version de HTTP utilisée ici. Par exemple, `[page]` peut valoir `/` pour demander la page d'accueil du site. `[version]` peut valoir :

- `1.0`, pour une requête puis fermeture de la connexion après réception de la réponse,
- `1.1`, qui permet en plus d'envoyer plusieurs requêtes à la suite.

On ajoute ensuite à la requête différents champs. On donne ci-dessous la description des champs essentiels.
On précise au serveur l'adresse de noms et le port correspondant à la page demandée en complétant la requête avec le message :

```
Host: [hostname]:443\r\n
```

où `[hostname]` est par exemple `www.wikipedia.fr`
On peut ensuite également demander au serveur de :

- attendre une nouvelle requête après cet échange avec le message « `Connection: keep-alive\r\n` » (fonctionnel uniquement en HTTP/1.1)
- ou terminer après un échange avec le message « `Connection: close\r\n` ».

Enfin, on dit au serveur avec quelle application on se connecte. Par exemple si notre application se nomme `client_https_simple`, alors le message est :

```
User-Agent: client_https_simple\r\n
```

La requête doit enfin terminer par une ligne vide (« `\r\n` »).

Réponse du serveur Une fois la requête envoyée par le client, celui-ci attend la réponse du serveur. La réponse est composée d'un entête, suivi d'une ligne vide (« `\r\n` »), suivie des données. La fin des données contenues dans la réponse ne se détecte malheureusement pas toujours de la même façon ! En effet les données peuvent finir par une ligne vide, ou bien l'entête peut contenir un champs `content-length` (ou `Content-length` ou `Content-Length` ou `content-Length`) qui donne la longueur en octets des données. Pour simplifier, on demande de ne gérer que les données terminant par une ligne vide.
Si la requête du client est bien formée, la première ligne de l'entête de la réponse doit être « `HTTP/1.0 200 OK` » ou « `HTTP/1.1 200 OK` » selon le protocole utilisé.
Pour détecter la fin de l'entête et la fin du message de la réponse, il suffit donc de chercher la suite de caractères « `\r\n\r\n` ».

1. Écrire un client sécurisé pour charger la page d'accueil d'un site web dont l'adresse de nom est donnée en argument de la ligne de commande.

   Le client ne faisant qu'un seul échange avec le serveur, peut détecter la fin d'envoi de la réponse du serveur lorsque ce dernier ferme la connexion.

   Le réseau de l'UFR étant protégé par un pare-feu, vous ne pouvez pas tester votre client depuis ce réseau. Testez-le sur votre machine personnelle reliée au wifi `up7d`, `eduroam` ou à un réseau de votre opérateur personnel.

2. Modifiez votre client afin qu'il puisse adresser plusieurs requêtes d'affilée au site web. Les demandes de pages du site se feront via la ligne de commande. Lorsque le client entre la chaîne « `quit` », le client ferme la connexion et termine.

   Vous devez impérativement, ici, utiliser le protocole HTTP/1.1 puisque votre connexion devra rester pérenne. Par ailleurs, pour chaque requête envoyée au serveur, il vous faudra détecter la fin de sa réponse. Le serveur ne fermant plus la connexion après l'envoi de sa réponse, le client doit donc procéder autrement :

   - il commence par lire tout l'entête (jusqu'à trouver la chaîne « `\r\n\r\n` »),
   - vérifie que la première ligne de cet entête est bien « `HTTP/1.0 200 OK` » et si ce n'est pas le cas, passe à la demande de requête suivante,
   - lit ensuite les données du message (jusqu'à trouver la chaîne « `\r\n\r\n` »).

   Attention, on rappelle qu'avec une connexion TCP, il faut gérer un flux de données.

Par exemple, si l'application client affiche un « `?` » pour signifier qu'il attend la référence de la page du site à télécharger, on veut obtenir le résultat suivant :

```
$ ./client_https_simple www.ietf.org
? /
[affichage de la réponse du serveur : entête + code html de la page d'accueil]
? /meeting/123/
[affichage de la réponse du serveur : entête + code html de la page de la rencontre]
? /triple/andouille/
[affichage de la réponse du serveur : entête commençant par "HTTP/1.1 404 Not Found"]
? quit
$
```