PR6 – Programmation réseaux
TP n° 7 : Un premier serveur web

Le protocole HTTP permet, entre autres, à un client de recevoir des données correspondant à certaines requête d'un serveur. Ce protocole est en particulier utilisé pour transmettre aux navigateurs (Firefox, Chrome, etc.) les documents HTML qui décrivent les pages web des sites Internet. Il 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 petit serveur HTTP en Java qui ne traitera que des requêtes de type `GET`.

Exercice 1 : Requêtes `GET`

Dans cet exercice, on va se familiariser avec HTTP et les requêtes `GET`, qui servent notamment à récupérer les pages des sites web.

1. Lancer `nc` en écoute sur le port 8080 avec la commande `nc -l 8080`. Avec un navigateur (Firefox ou Chrome), visiter la page `http://localhost:8080/`. Que se passe-t-il au niveau de `nc` ? Et au niveau du navigateur ?
2. Le message envoyé à `nc` contient une ligne de la forme `"GET [page] HTTP/[version]"` où `[page]` est la page demandée par la requête et `[version]` la version de HTTP utilisée ici. Quelle est la page demandée ainsi que la version de HTTP ici ?
3. La suite du message contient des lignes de la forme `"[Header]: [contenu]"` qui paramètrent la requête ou donnent plus d'informations sur l'émetteur de cette requête. Que contiennent les en-têtes `Host` et `User-Agent` dans votre cas ?
4. Recommencer la procédure en allant visiter cette fois la page `http://localhost:8080/dossier/index.html`. Comment est modifiée la ligne de requête `"GET ..."` ?
5. (Si vous avez Firefox) Envoyer du texte aléatoire au niveau de `nc` puis clore la communication. Que se passe-t-il au niveau du navigateur ?

On peut voir une réponse normale à un `GET` en faisant une requête à un vrai serveur HTTP. Par exemple, en lançant la commande `nc -C www.google.fr http` et en entrant la requête

```
GET / HTTP/1.0
```

suivies de deux retours à la ligne, on obtient une réponse qui ressemble à celle-ci :

```
HTTP/1.0 200 OK
Date: Tue, 02 Mar 2021 16:33:06 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Accept-Ranges: none
Vary: Accept-Encoding

<!doctype html><html><head>...
```

Malheureusement, cette procédure ne fonctionne pas sur `lulu` à cause du pare-feu que cette machine utilise, mais essayez-la si vous êtes sur une autre machine ! Ainsi, la première ligne d'une réponse à une requête HTTP est de la forme

```
HTTP/[version] [code] [message]
```

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>Ma page web !</title>
    </head>
    <body>
        <p> Ceci est un paragraphe de ma page web. </p>
    </body>
</html>
```

FIGURE 1 – Un exemple de page HTML

où `[version]` est la version d'HTTP utilisée, `[code]` est un nombre qui précise si la requête a réussi ou échoué (les codes en `2XX` indiquent un succès) et où `[message]` est un message associé au code précédent. Les lignes suivantes sont des lignes de *headers* de la forme `"[Header]: [contenu]"`. Par exemple, on a ici un *header* `Date` qui précise quand a été faite la requête. On a aussi un header `Content-Type` qui précise le type de document envoyé (ici, HTML). Se trouve ensuite une ligne vide, avant que ne commence le document HTML envoyé (la page d'accueil de Google).

Avant de passer à l'écriture d'un serveur HTTP dans le dernier exercice, on peut auparavant faire l'exercice suivant pour se (re)familiariser avec le HTML.

Exercice 2 : Fichiers HTML

(Si vous connaissez déjà le HTML, vous pouvez passer à la suite). Un document HTML est un fichier essentiellement constitué d'une arborescence de blocs délimités par des balises de la forme `<nom> ... </nom>`. Parmi ces balises, on distingue :

- le bloc `<html> ... </html>` qui est le bloc « racine » qui contient tous les autres ;
- le bloc `<head> ... </head>` qui contient des méta-données de la page ;
- le bloc `<body> ... </body>` qui contient le contenu à afficher de la page.

Pour le contenu de la page, on peut utiliser les blocs `"<p> ... </p>"` pour faire des paragraphes. Un exemple simple de document HTML est donné par la Figure 1.

1. Créer un fichier `.html` avec le code HTML donné par la Figure 1 et l'ouvrir avec un navigateur. Est-ce que ce qui s'affiche est cohérent avec le contenu du document ?
2. Aller sur la page `https://www.irif.fr/~sangnier/enseignement/reseaux.html` et regarder son code source (Clic droit, puis « Afficher le code source »). Observer rapidement les différentes balises utilisées pour les différents éléments de la page. Comment sont représentés les caractères spéciaux comme les accents ?

Exercice 3 : Serveur

Comme vu plus haut, une requête HTTP de type `GET` contient une première ligne de la forme

```
GET [chemin] HTTP/[version]
```

suivie d'un certain nombre de lignes de *headers* terminées par une ligne vide. Chaque « ligne » se termine en fait par les caractères `<CR><LF>` (ou `\r\n` en C et Java).

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Connection: close

<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>Ma page web !</title>
    </head>
    <body>
        <p> Ceci est un paragraphe de ma page web. </p>
    </body>
</html>
```

FIGURE 2 – La réponse à renvoyer au client

1. Faire un serveur en Java qui écoute sur le port 8080. Pour chaque client qui se connecte, le serveur lira les données envoyées par le client jusqu'à tomber sur la séquence `\r\n\r\n` (une ligne vide). À ce moment, le serveur renverra la réponse de la Figure 2. Tester l'affichage de la page web avec un navigateur.
2. Utiliser des threads pour permettre de traiter différents clients en parallèle.
3. Dans la page HTML renvoyée, ajouter un message « Vous êtes la `[n]`ème personne à vous connecter ici » où `[n]` sera un compteur incrémenté par les threads gérant les clients. Ne pas oublier d'utiliser des verrous pour protéger la modification de ce compteur.
4. Ajouter à la page HTML renvoyée au client un paragraphe qui contient la requête HTTP entière faite par le client (en HTML, les retours à la ligne se font par la balise `<br>`).
5. Parser la requête du client afin de récupérer la page demandée par `GET`, ainsi que le nom et le contenu des différents *headers*. On utilisera ces données pour afficher un paragraphe supplémentaire sur la page HTML de la forme « Vous avez visité le lien `[lien]` avec la version `[version]` de HTTP », où
   - `[lien]` sera la concaténation `[host] + / + [chemin]` où `[host]` est le contenu du *header* `Host` et `[chemin]` est l'argument de `GET` ;
   - `[version]` sera la version de HTTP indiquée dans la requête.

   Ajouter aussi un message indiquant le navigateur utilisé par le client (qui est donné par le contenu du *header* `User-Agent`).
6. Si le chemin demandé par `GET` est de la forme `/[page].html` et qu'il existe un fichier `[page].html` dans le dossier courant, renvoyer le contenu de ce fichier au client. Tester ce comportement en créant plusieurs fichiers HTML dans le dossier courant à partir du modèle donné en Figure 1.