TP3 : HTTP

1 Fonctionnement

HTTP est un protocole client-serveur basé sur TCP et permettant (à l'origine) l'échange de fichiers texte. Le client (navigateur Web) :

- Cherche des données (analyse de l'URL, résolution DNS, connexion TCP au serveur, formulation d'une requête, réception des données)
- Met en forme (interprétation de l'HTML, récupération de fichiers CSS/js/images)

Le serveur Web écoute les requêtes, vérifie leur validité et répond avec des données, un message d'erreur, une demande d'authentification,...

À l'origine, une requête correspondait à une connexion. En particulier, le serveur ne conserve pas d'information sur les requêtes précédentes.

1.1 Communication

Plusieurs types de requêtes existent, mais la plus utilisée est `GET`, qui demande les données associées à une certaine URL. Cette requête est de la forme

```
GET <url> HTTP/<version>
```

suivie d'une ligne vide.

Les réponses du serveur sont de la forme

```
HTTP/<version> <code> <message>
```

suivie d'une ligne vide, puis le contenu de la page (si la requête aboutit).

Chaque requête peut contenir des informations supplémentaires, sous la forme `Champ:valeur` (un par ligne). Par exemple, on peut spécifier l'encodage des données, leur date de dernière modification, le type des fichiers, la taille de la réponse, ...

Dans votre navigateur favori, affichez les requêtes HTTP envoyées. Accédez à l'URL `http://perdu.com` et déterminez quels champs correspondent aux propriétés énoncées ci-dessus.

Autres requêtes :

- `HEAD` permet d'obtenir seulement l'en-tête de la réponse.
- `POST` permet de remplir des formulaires.
- `PUT` permet d'envoyer une page au serveur (l'inverse de GET).
- `DELETE` permet de supprimer une page.
- `TRACE` sert au debug.
- `OPTIONS` fournit des informations sur les requêtes supportées par le serveur.

1.2 Réponses et erreurs

Chaque réponse du serveur est précédée d'un code. Comme pour FTP, il s'agit de 3 chiffres, dont le premier indique le type de la réponse:

- 1xx : requête reçue, traitement en cours
- 2xx : succès
- 3xx : redirection
- 4xx : erreur du client
- 5xx : erreur du serveur

Quels sont les messages associés aux codes 200, 404, 418 et 502 ?

2 Communication à la main

Vérifier que la commande `telnet` fonctionne bien dans votre terminal (sous windows, il faut parfois aller l'activer dans le panneau de configuration).

Utiliser la commande `telnet perdu.com http` pour vous y connecter, puis envoyer les requêtes suivantes :

- `GET /\r\n`
- `GET / HTTP/1.1\r\n\r\n`
- `GET / HTTP/1.1\r\nHost:perdu.com\r\n\r\n`

Dans chaque cas, expliquer la réponse obtenue. À quoi sert le champ `Host` ?

3 Mini-site web

Le but de ce TP est de créer un serveur HTTP qui pourra répondre à des requêtes GET. Afin de le tester, créez d'abord un dossier contenant un mini-site web de quelques pages HTML statiques réparties dans plusieurs sous-dossiers, et contenant des liens entre elles.

Si vous êtes en manque d'inspiration, vous pouvez partir de la page du cours et enlever les balises correspondant aux images.

4 Serveur HTTP simplifié

Architecture

On rappelle que le protocole HTTP fonctionne sur TCP, et que le serveur doit pouvoir traiter les requêtes de plusieurs clients en parallèle. Implémentez l'architecture de base de ce type de serveur (vous pouvez vous aider du TP1). Dans un premier temps, le serveur affichera les requêtes émises sur sa sortie standard (aucune vérification de la forme des requêtes n'est faite à ce stade). Pensez à bien gérer les exceptions. Testez votre serveur via telnet, puis en vous connectant à `localhost` depuis votre navigateur.

Décodage des requêtes HTTP

Dans cette partie on veut écrire des méthodes qui permettent la réception et l'analyse de requêtes. Dans ce TP, on ne traite que les requêtes GET, il faudra répondre à toutes les autres par une erreur 405 (Méthode non prise en charge).

Pour rappel, le format d'une requête GET est le suivant :

- `GET <chemin> HTTP/<version>` : on ne considère ici que les versions 1.0 et 1.1, et des chemins absolus.
- Un certain nombre de lignes `clé:valeur`. Le serveur ignorera tous les champs qui ne sont pas `Host:` (voir dernière section).
- Une ligne vide.
- Un message (que le serveur ignorera).
- Une ligne vide.

Chaque ligne termine par `\r\n`.

Écrire une méthode qui reçoit une requête HTTP (quelle qu'elle soit) et la stocke sous forme de chaîne de caractères. Écrire ensuite une méthode qui vérifie si une requête est bien formée. Cette méthode renverra un code HTTP : 400 (Requête mal formée), 405 s'il ne s'agit pas d'un GET, et 200 si tout va bien.

Pour vous aider dans l'analyse de chaînes de caractères, vous pouvez vous aider de la bibliothèque `java.util.regex`.

Pensez à tester ces deux méthodes !

Réponses du serveur – les erreurs

Le format des réponses du serveur est le suivant :

- `HTTP/<version> <code> <message>` où code est un nombre à 3 chiffres.
- Un certain nombre de lignes `clé:valeur`.
- Une ligne vide.
- Un message (par exemple le contenu du fichier quand la requête aboutit).
- Une ligne vide.

Écrire une méthode (surchargée) qui prend en argument soit un code, soit un code et une taille, et qui génère l'en-tête d'une réponse du serveur (c'est-à-dire tout jusqu'à la première ligne vide incluse). Cet en-tête comportera :

- le code et le message associé (200 → OK, 400 → BAD REQUEST, 404 → NOT FOUND, 405 → METHOD NOT ALLOWED, 500 → INTERNAL SERVER ERROR). Vous pouvez stocker le message associé à un code dans un dictionnaire, et renvoyer une erreur 500 si le dictionnaire ne contient pas la clé qu'on y cherche.
- Un champ `Date:` suivi de la date au format suivant : `lun., 29 nov.  2020 09:45:39 CET`.

- Un champ `Server:<nom de votre serveur>`.
- Un champ `Connection:close` qui indique que la connexion se ferme à la fin du traitement de la requête (pas de keep-alive).
- Un champ `Content-Length:<taille>` contenant la taille (en octets) des données envoyées (hors en-tête), seulement si cet argument est présent.
- Un champ `Content-Type:<type>` dont la valeur sera toujours (pour le moment) `text/html; charset=utf-8`.
- Une ligne vide.

Écrire ensuite une méthode qui génère les réponses lorsqu'une erreur se produit. Il s'agit de rajouter un message (par exemple le code HTML d'une page d'erreur) et une ligne vide à la chaîne générée par la méthode précédente. À nouveau, vous pouvez utiliser un dictionnaire pour stocker le message associé à chaque erreur.

Réponses du serveur – les vraies

Écrire une méthode prenant en entrée une requête, et qui construit le chemin vers le fichier demandé. Par exemple, si votre serveur est situé dans le répertoire `/chemin/du/site/`, et que l'adresse fournie par la requête est `/chemin/de/la/requete`, la méthode renverra `/chemin/du/site/chemin/de/la/requete`.

Deux remarques :

- si le chemin présent dans la requête mène à un dossier et non à un fichier, la méthode doit renvoyer le chemin vers le fichier `index.html` présent dans ce dossier. Dans l'exemple précédent, si `/chemin/du/site/chemin/de/la/requete` est un dossier, la méthode doit renvoyer `/chemin/du/site/chemin/de/la/requete/index.html`.
- Le champ de la requête peut être pollué : il faut éliminer tout ce qui se trouve après un éventuel `?` (inclus), et remplacer les `%xx` par le caractère ASCII numéro `xx`.

Écrire (enfin !) une méthode qui génère la réponse au serveur quand il n'y a pas d'erreur. Vous devriez alors avoir un serveur fonctionnel : testez le avec votre navigateur !

Extension : virtualisation

Le serveur, dans son état actuel, gère un site web situé sur votre machine dans le dossier que vous avez choisi, disons `/chemin/du/site`. Il est tout à fait possible (c'est même répandu) qu'une même machine héberge plusieurs serveurs.

Modifier votre serveur pour qu'il prenne en charge deux sites web (situés dans deux dossiers distincts). Il faudra alors se baser sur le champ `Host:` pour savoir à quel site le client essaye de se connecter, et renvoyer une erreur quand ce champ sera absent.