Examen de « Programmation Réseau »
Licence Informatique – Université Paris Diderot

Jean-Baptiste Yunès

20 Juin 2013

On se propose d'écrire un mini serveur web en Java. On rappelle que sur le web il a deux interlocuteurs : le navigateur (partie cliente) et le serveur. Pour obtenir une « page » le client utilise une URL ; par exemple `http://www.machintruc.com/bidule/toto.html`. Ce qui a pour effet de déclencher auprès du serveur web en attente en TCP sur la machine `www.machintruc.com` et sur le port par défaut `80`, un dialogue tel que décrit par le protocole HTTP. Le client envoie d'abord un ensemble de lignes de texte terminé par une simple ligne blanche. Les lignes envoyées doivent être au moins deux. La première et nécessairement (pour l'URL donnée en exemple) `GET /bidule/toto.html HTTP/1.0`. La seconde `Host: www.machintruc.com`. On ne décrit pas ici les autres lignes sachant que le serveur est autorisé à les ignorer. En réponse à une telle demande le serveur renvoie un message. Un message de réponse consiste d'abord en une ligne décrivant le type de réponse (erreur ou non), puis par des lignes d'entête, suivies par une ligne vide, suivie par un contenu. Les types de réponses qui nous intéressent sont :

- `HTTP/1.1 200 OK` pour signifier que la requête peut être servie normalement.
- `HTTP/1.1 403 Forbidden` pour indiquer que le document existe mais est interdit d'accès.
- `HTTP/1.1 404 Not Found` pour indiquer que le document demandé n'existe pas.
- `HTTP/1.1 500 Internal Server Error` pour indiquer que le serveur ne peut pas répondre normalement pour des raisons techniques exceptionnelles.

Les lignes d'entêtes que notre serveur renverra sont :

- `Server: nom du serveur`
- `Content-Length:` *taille*, la taille en octets du contenu qui suivra la ligne blanche

En cas d'erreur, le serveur renverra toujours un contenu non nul (une page web prédéfinie avec un message d'erreur). Par exemple en réponse à la demande de chargement de l'URL `http://zigoto.fr/abracadabra.html`, le serveur répond (car le document n'existe pas) :

```
HTTP/1.1 404 Not Found
Server: Apache
Content-Length: 221

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /abracadabra.html was not found on this server.</p>
</body></html>
```

Lorsqu'une connexion est créée, le serveur tentera de répondre à chaque requête envoyée par le client, mais il coupera la connexion automatiquement lorsque qu'un délai trop long (2 minutes) se sera écoulé depuis la première requête reçue. Le serveur sera multithreadé.

1. écrire le `main` de la classe `MyHTTPD` qui lorsque exécutée par la commande `java MyHTTPD` *port* *nbmax*, attends sur une connexion TCP sur le port indiqué et appelle une fonction statique de nom `service` prenant en paramètre la `Socket` de service obtenue.

2. on souhaite faire en sorte que le serveur n'autorise pas plus de $nbmax$ connexions simultanées. Pour cela, on emploiera un compteur (une variable statique de type `int` qui devra être manipulé par différentes fonctions statiques `addOneService`, `removeOneService`. Ces fonctions seront appelées par chaque `Thread` lorsqu'il démarre et lorsqu'il s'arrête.
   (a) quel(s) problème(s) la manipulation de ce compteur pose t'il ?
   (b) comment y remédier ?
   (c) écrire les fonctions `addOneService`, `removeOneService`.
3. écrire la fonction `service` qui, si le nombre de connexions est dépassé renvoie au client une erreur de type 500 avec un message adéquat (appel à la fonction `void send500(Socket s,String message);` et qui si le nombre n'est pas dépassé lance un service, c'est-à-dire un `Thread` attaché à une instance d'une classe `HTTPService` (implémentation de l'interface `Runnable`) qui aura comme attribut la `Socket` de service.
4. écrire la méthode `run` de la classe `HTTPService` qui, dans une boucle infinie, décode le message du client (ligne de requête et identification du serveur) et envoie une réponse adéquate : 403 si le document existe mais n'est pas accessible, 404 si le document n'existe pas et 200 s'il existe et est accessible. La réponse est en fait envoyée via des méthodes ayant pour signatures `void send403(Socket s,String message);`,`void send404(Socket s,String message);` et `void sendFile(Socket s,File f);`. Le décodage consiste à appeler une méthode statique de la classe `MyHTTPD` de signature `File decode(String pathname);`. On utilisera les méthodes adéquates des instances de la classe `File`, c'est-à-dire `boolean exists();`,
   `boolean canRead()`. La classe `File` possède un constructeur à un argument de type `String`.
5. écrire la méthode de signature `void send404(Socket s,String message);` qui envoie la ligne de réponse, les lignes d'entête et le contenu correspondant (voir exemple en introduction).
6. écrire la méthode de signature `void send403(Socket s,String message);` qui envoie la ligne de réponse, les lignes d'entête et le contenu correspondant (s'inspirer de l'exemple en introduction).
7. écrire la méthode de signature `void send500(Socket s,String message);` qui envoie la ligne de réponse, les lignes d'entête et le contenu correspondant (s'inspirer de l'exemple en introduction).
8. écrire une méthode de signature `void sendFile(Socket s,File f);` permettant d'envoyer la ligne de réponse, les lignes d'entête et le contenu associé au fichier sur la socket de service.
9. ajouter la fonctionnalité permettant d'arrêter le service lorsqu'un délai trop grand c'est écoulé entre la réception de la première requête et la nouvelle.
   (a) décrire le mécanisme envisagé. Le critiquer.
   (b) écrire le code correspondant.
10. quel(s) problème(s) de sécurité ce serveur possède t-il ? Expliquer. Comment le(s) corriger ?