Programmation réseaux

TP 5 – Client/serveur TCP en Java: HTTP

Mars 2007

Introduction

Lors du TD1, nous avons rencontré un exemple de protocole basé sur TCP, à savoir SMTP. L'objet de ce TD est de revenir plus en détail sur la programmation TCP en Java, via l'implémentation d'un client et surtout d'un mini-serveur HTTP.

Le protocole HTTP (HyperText Transport Protocol) est le protocole utilisé pour la communication de documents pour le World Wide Web. Il permet la transmission de documents HTML, d'images ou autres entre un serveur web et un navigateur. HTTP est défini dans la RFC 2616 (176 pages cette fois-ci).

1 Le protocole

1.1 Format des requêtes

Une requête HTTP minimale est un message textuel de deux lignes ayant la forme suivante :

```
GET / HTTP/1.0
```

Cette requête demande le fichier racine `/` du serveur contacté, en utilisant la variante 1.0 de la norme HTTP. Attention, cette norme spécifie que les retours à la lignes doivent être marqués par des `"\r\n"` (encore que de simples `"\n"` semblent être tolérés par certains serveurs).

Plus généralement :

- la méthode `GET` de l'exemple peut être remplacée par d'autres méthodes. La principale autre est `POST`, qui ne diffère de `GET` que par la façon de transmettre les arguments des formulaires, dont nous ne nous occuperons pas ici.
- A la place du premier `/`, peut se trouver le chemin menant au fichier voulu sur le serveur, du genre `/c/est/ici/truc.html`, voire même l'URL complète du fichier voulu, comme `http://www.foo.fr/c/est/ici/truc.html`.
- Entre la ligne initiale et la ligne vide, peuvent se trouver des lignes facultatives précisant par exemple le navigateur utilisé (`User-Agent: XXX`), la machine
- Pour information, dans le cas de `POST`, la ligne vide ne termine pas forcement la requête, les arguments `POST` pouvant venir par la suite.

1.2 Format des réponses

La réponse du serveur est constituée d'un entête et d'un corps, séparés par une ligne vide :

- L'entête commence par une ligne de statut, du genre `HTTP/1.1 200 OK`. Après la norme utilisée par le serveur[1], on trouve un code indiquant l'issue de la transaction, et son explication en anglais. Un code de la forme `2xx` marque un succès, `3xx` correspond à une

[1] La variante 1.1 apparaîtra peut-être dans les réponses, elle est plus récente et étend légèrement 1.0, sans que cela importe pour ce TD

redirection, `4xx` à une erreur du coté client (dont le fameux `404`), et `5xx` à une erreur interne au serveur.
- Après cette ligne de statut se trouver un certain nombre de lignes indiquant des informations telles que l'heure, le type et la version du serveur, le type du fichier retourné (`Content-Type : ...`), sa taille (`Content-Length : ...`), etc.
- En cas de succès de la requête, après la première ligne vide, on trouve le corps du message de réponse, constitué du fichier demandé.

2 Primitives Java utiles

La classe Socket

- `Socket(InetAddress addr, int port)`
  ouvre une connection TCP et construit un objet `Socket` associé.
- `void close()`
- `OutputStream getOutputStream()`
- `InputStream getInputStream()`

La classe ServerSocket

- `ServerSocket(int p)`
  construit une socket de serveur destinée à écouter sur le port local `p`.
- `Socket accept()`
  attend la connection d'un client et retourne alors une socket permettant la communication avec ce client.

Entrées/Sorties

Un objet `PrintStream` étend `OutputStream` avec les primitives usuelles `print`, `println`, etc. De même, un `InputStream` est assez limité : `read()` pour la lecture d'un caractère, ou bien `read(byte [] b)` pour la lecture d'un tableau de caractères. Il peut alors être plus commode de passer à une structure `BufferedReader` qui fournit en particulier une méthode `readLine()`. Cela se fait via l'incantation suivante :

```
InputStream is = ...;
BufferedReader br = new BufferedReader(new InputStreamReader(is));
```

Exercice 1 – *Client HTTP simple*

Écrivez un programme Java qui se connecte en TCP à un serveur HTTP, demande une page HTML et affiche sur sa sortie standard le texte HTML reçu. Si vous avez eu le temps de traiter l'exercice 3.1 du TD1, vous pouvez vous inspirer de votre client SMTP.

Exercice 2 – *Serveur HTTP Simple*

Écrivez un programme attendant une connection sur un port donné, capable de recevoir des requêtes HTTP, et de toujours renvoyer une même réponse HTML minimale pour toutes les requêtes possible.

Testez votre serveur en utilisant un navigateur web (netscape ou plus léger : lynx, wget ...).
Attention : le port sur lequel votre programme écoutera devra être :
- supérieur à 1024, vu que vous n'avez pas les droits d'administrateur

– disponible, et en particulier distinct des ports utilisés par vos collègues.

Exercice 3 – *Serveur HTTP multi-thread*

Dans la réalité, la reception d'une requête, son traitement et l'envoi d'une réponse peut prendre un temps non négligeable, comme par exemple dans le cas de pages dynamiques (Cf. PHP-MySQL par exemple). Simulez ce temps de réponse à l'aide d'une temporisation. Que se passe-t'il si une requête arrive pendant une telle temporisation.

Pour être toujours en mesure d'accepter de nouvelles requêtes, modifiez votre serveur pour que chaque requête reçue soit traitée par une nouvelle thread. Faites afficher régulièrement par votre serveur le nombre de thread en fonctionnement.

Exercice 4 – *Protection contre certains dénis de service*

Une attaque possible contre un serveur web est de le saturer de nouvelles connections sans jamais transmettre les requêtes proprement dites. Un serveur naïf tel que celui de l'exercice précédent risque alors d'atteindre le nombre maximal de threads autorisées, empêchant le traitement des requêtes légitimes : on parle alors de déni de service (DoS).

Comment faire pour que chaque thread ne dépasse pas une certaine durée de vie et/ou termine s'il ne reçoit pas rapidement une requête complète ?

Extensions possibles côté client :

- dans l'affichage, distinguer l'entête HTTP du corps du message, par exemple via les préfixes `ENTETE` et `CORPS`.
- "embellir" l'HTML retourné, par exemple en n'affichant pas les balises HTML `<...>`.

Extensions possibles côté serveur :

- Enrichisser le message répondu par votre serveur : afficher par exemple l'heure, le nom de la machine serveur, son système d'exploitation, la version de Java utilisée, ou encore des informations sur le client : numéro IP, navigateur, fichier demandé.
- Une fois décodé le nom de fichier demandé, tenter d'adapter la réponse en conséquence, soit en accédant au système de fichiers sur le serveur, soit en générant une page au vol, par exemple dans le style de `http://www.eleves.ens.fr/infinity/`