UPMC
SORBONNE UNIVERSITÉS

SRCS : Systèmes Répartis Client/Serveur

TD 02 – Programmation client/serveur en mode message

Janvier 2016

Exercice 1 : Serveur Web

Le but de cet exercice est d'implanter une partie du protocole HTTP/1.0 (commandes `GET` et `PUT`) avec l'API socket du langage Java. Ce TD sert de préparation aux TP 3 et 4. API socket Java

Question 1

Quelle méthode Java permet d'attendre l'arrivée de demandes de connexions sur un port TCP ?

Question 2

Comment lire des données sur une socket en Java ?

Question 3

Comment écrire des données sur une socket en Java ?

Question 4

A partir de l'extrait de la RFC décrivant le protocole *HTTP* fourni en annexe, quel est le rôle de la commande `GET` ? Donner le format de la commande et préciser ce qu'envoient et reçoivent le client et le serveur ?

Question 5

Ecrire le pseudo-code d'un serveur *HTTP* gérant la commande `GET`.

Question 6

Proposer une solution pour que le serveur puisse gérer simultanément plusieurs clients émettant des requêtes `GET`.

Question 7

La commande `PUT` permet à un client Web (navigateur Internet ou programme quelconque), d'envoyer à distance sur un serveur Web un fichier. De même que la commande FTP `PUT`, la commande *HTTP* `PUT` est une commande d'écriture d'un fichier sur un serveur (alors que `GET` est une commande de lecture). La commande `PUT` fait partie des extensions introduites par la version 1.1 du protocole *HTTP* 1.1.

Le client envoie la commande suivante :

```
PUT nomDeFichier versionHTTP
   en-tête1: valeur1
   ...
   en-têten: valeurn
   ligne blanche
   Contenu du fichier
```

L'en-tête Content-Length est obligatoire et fourni la taille des données envoyées. Par exemple :

```
PUT /index.html HTTP/1.1
   Content-Length: 10
   ligne blanche
   xxxyyyyzzz
```

Le serveur reçoit une telle commande. Si le fichier ne peut pas être écrit ou s'il y a une autre erreur (par exemple commande *HTTP* mal formée), il envoie un message d'erreur de la forme :

```
HTTP/version codeErreur signification
ligne blanche
Message d'erreur
```

Par exemple :

```
HTTP/1.1 404 Not Found
ligne blanche
Fichier inexistant
```

Sinon, si le fichier existe, il envoie un message de la forme :

```
HTTP/version 200 OK
en-tête1: valeur1
...
en-têten: valeurn
```

Par exemple :

```
HTTP/version 200 OK
```

Pourquoi l'en-tête Content-Length est-elle obligatoire ?

Question 8

Modifier le pseudo-code de la question 5 pour ajouter au serveur *HTTP* la gestion d'une commande `PUT`.

Question 9

Donner le pseudo-code d'un client Web envoyant une commande `PUT` à un serveur.

Question 10

Quel problème nouveau est introduit par la commande `PUT` ?

Question 11

Proposer une solution.

Exercice 2 : Serveur calculette

En utilisant un protocole de transport fiable (TCP), on souhaite réaliser un serveur qui implante les fonctionnalités d'une calculatrice élémentaire fournissant quatre opérations (+ - * /) . Des clients doivent donc pouvoir se connecter à distance et demander au serveur la réalisation d'une opération. Le serveur leur répond en fournissant le résultat ou un compte rendu d'erreur.

Question 1

Définir un protocole applicatif (i.e. un ensemble de messages et leur enchaînement) aussi simple que possible implantant cette spécification.

Question 2

Recenser les cas d'erreur pouvant se produire avec ce protocole applicatif et proposer un comportement à adopter lorsque ces erreurs surviennent (on ne s'intéresse ici ni aux erreurs de transport du style message perdu, ni aux pannes de machines).

Question 3

En utilisant les primitives fournies ci-dessous donner le pseudo-code du serveur et le pseudo-code d'un client demandant la réalisation de l'opération 8.6*1.75 :

ouvrirCx(serveur) : demande d'ouverture de connexion vers serveur

attendreCx() : attente bloquante et acceptation d'une demande de connexion

envoyer(données) : envoi de données

recevoir() : attente bloquante et réception de données

fermerCx() : fermeture de connexion

Question 4

Le serveur est-il avec ou sans état ? Remarque : on se place au niveau des primitives définies à la question précédente. On ne s'intéresse pas au protocole de transport sous-jacent.

Question 5

Si vous avez répondu sans état à la question précédente, donner un exemple de serveur calculette avec état. Réciproquement, si vous avez répondu avec état, donner un exemple de serveur calculette sans état.

Question 6

On souhaite que plusieurs clients puissent se connecter simultanément sur le serveur. Cela pose-t-il un problème particulier ? Modifier le pseudo-code du serveur pour prendre en compte ce cas.

Question 7

On souhaite maintenant que les clients puissent enchaîner plusieurs demandes d'opérations au sein d'une même connexion. Proposer deux solutions pour cela (une en modifiant le protocole précédent et une sans modification du protocole).

