Université Lille 1 Mai 2014
Master mention Informatique – M1
Module : Conception d'applications réparties (CAR)

Examen

Conception d'applications réparties (CAR)

Tous documents papier autorisés. Téléphones, calculatrices et ordinateurs interdits.
Le barème est donné à titre indicatif. 3 heures.
Ce sujet comporte 2 pages.

1 Protocole HTTP 2.0 (7 points)

1.1 Expliquer le principe de la notion de connexion persistante du protocole HTTP 1.1. (1 point)

On souhaite étudier différentes évolutions du protocole HTTP. On considère le cas du transfert avec la commande `GET` d'un fichier index.html faisant référence à un fichier style.css et un fichier image.png. On envisage trois solutions pour l'envoi de ces trois fichiers : (1) trois connexions TCP successives, contenant chacune un échange requête/réponse HTTP, (2) une connexion TCP contenant trois échanges requête/réponse HTTP successifs, (3) une connexion TCP contenant un seul échange requête/réponse HTTP.

1.2 Discuter brièvement des avantages et des inconvénients des trois solutions. (1 point)
1.3 Proposer une structure de message pour la requête et la réponse de la solution (3). (1 point)

On propose maintenant de remplacer les messages en mode caractère du protocole HTTP actuel par des messages **binaires**.

1.4 Proposer une structure de message binaire pour la requête et la réponse de la solution (3). (1 point)
1.5 Écrire en Java ou en pseudo-code le code d'un serveur mono-threadé correspondant à cette solution. (2 points)

On souhaite introduire un mode de communication dit *push* permettant au serveur de transmettre spontanément, sans que le client en ait fait la requête, des données au client. Cela peut correspondre par exemple à des fichiers image dont le serveur sait que le client va avoir besoin pour afficher la page HTML.

1.6 Proposer en français une modification du protocole HTTP permettant de mettre en œuvre le mode *push*. Quelle conséquence ce mode a-t-il sur le fonctionnement du client ? (1 point)

2 Protocole d'invocation de méthodes distantes (5 points)

Cet exercice a pour but d'étudier la mise en œuvre d'un protocole, que l'on nommera CARMP, qui est similaire au protocole JRMP de Java RMI, et qui permet à un client d'invoquer à distance les méthodes d'un objet serveur en utilisant TCP. On suppose que plusieurs objets serveurs peuvent utiliser le même port TCP. On souhaite que le client puisse détecter, entre l'invocation et son retour, les pannes de l'objet serveur en envoyant toutes les 10 secondes un message `ping` auquel l'objet serveur est tenu de répondre

par un message `pingack`. Si au bout de 3 envois de `ping`, le client n'a reçu aucun message `pingack`, il considère que l'objet serveur est en panne et lève une exception. Le protocole définit quatre messages : (1) `request` permet d'envoyer une invocation de méthode, (2) `reply` retour de l'invocation de méthode, (3) `ping` test de l'activité de l'objet serveur, (4) `pingack` réponse à `ping`.

2.1 Proposer une structure de données pour représenter la référence distante d'un objet serveur. (1 point)
2.2 Proposer une structure de données pour représenter les messages du protocole CARMP. (1 point)

On considère un client qui invoque la méthode suivante sur un objet serveur :

```
double[][] inverser_matrice( double[][] matrice );
```

2.3 Écrire en Java ou en pseudo-code le code du client qui utilise le protocole CARMP pour réaliser une invocation de la méthode `inverser_matrice`. (3 points)

3 Élection sur un graphe d'objets RMI (4 points)

On considère un nombre quelconque d'objets RMI reliés selon une topologie logique en graphe. Chaque objet connaît un tableau de voisins (variable `voisins`) avec qui il est capable de communiquer. Un identifiant unique (un entier) est attribué à chaque objet. L'élection est un mécanisme qui consiste à élire l'objet d'identifiant le plus élevé. L'élection est réalisée à l'aide d'une invocation qui parcourt le graphe. Elle peut être déclenchée par n'importe quel objet qui dans ce cas, s'appelle un initiateur. À la fin de l'élection, tous les objets du graphe doivent connaître l'identifiant de l'élu.

3.1 Proposer une interface Java RMI pour les objets du graphe. (0,5 point)

3.2 Écrire en Java ou en pseudo-code le code de la classe mettant en œuvre cette interface. (2,5 points)

3.3 Plusieurs parcours peuvent-ils être déclenchés concurremment sur le graphe avec votre algorithme ? Si oui pourquoi ? Si non, proposer, en français une solution pour cela. (1 point)

4 Application Twitter-*like* (4 points)

On souhaite mettre en place une application Java EE de type Twitter, permettant de gérer des utilisateurs et des messages écrits par ces utilisateurs. Chaque utilisateur (`User`) a un identifiant unique (de type entier) et un nom (chaîne de caractères). Chaque message (`Message`) est horodaté (variable de type `java.util.Date`), possède un identifiant unique (entier) et un contenu (chaîne de caractères). Un message est associé à un utilisateur qui peut avoir zéro ou plusieurs messages. L'interface d'accès à l'application, basée sur REST, fournit huit méthodes, `addUser`, `removeUser`, `getUser`, `modifyUser`, `addMessage`, `removeMessage`, `getMessage`, `modifyMessage`, pour respectivement, ajouter, supprimer, récupérer, modifier un utilisateur et de même pour un message.

4.1 Quel(s) type(s) de *bean*(s) session et/ou entité proposez-vous de définir pour mettre en œuvre cette application ? (1 point)

4.2 Écrire le code Java des classes correspondantes. (2 points)

On suppose maintenant que les utilisateurs ont des abonnés (*followers*) et des abonnements (*following*).

4.3 Décrire en français les modifications que vous proposez pour mettre en œuvre cela. (1 point)